UNIVERSIDADE DO EXTREMO SUL CATARINENSE - UNESC

CURSO DE CIÊNCIA DA COMPUTAÇÃO

FABRICIO MEDEIROS SOMINI

PROPOSTA PARA MELHORIA NA CRIAÇÃO DE TESTES AUTOMATIZADOS UTILIZANDO OS REGISTROS DAS ATIVIDADES DOS USUÁRIOS DE APLICAÇÕES MÓVEIS

CRICIÚMA

2018

FABRICIO MEDEIROS SOMINI

PROPOSTA PARA MELHORIA NA CRIAÇÃO DE TESTES AUTOMATIZADOS UTILIZANDO OS REGISTROS DAS ATIVIDADES DOS USUÁRIOS DE APLICAÇÕES MÓVEIS

Trabalho de Conclusão de Curso, apresentado para obtenção do grau de Bacharel no curso de Ciência da Computação da Universidade do Extremo Sul Catarinense, UNESC.

Orientadora: Prof.ª MSc. Ana Claudia Garcia Barbosa

CRICIÚMA

2018

FABRICIO MEDEIROS SOMINI

PROPOSTA PARA MELHORIA NA CRIAÇÃO DE TESTES AUTOMATIZADOS UTILIZANDO OS REGISTROS DAS ATIVIDADES DOS USUÁRIOS DE APLICAÇÕES MÓVEIS

Trabalho de Conclusão de Curso aprovado pela Banca Examinadora para obtenção do Grau de Bacharel, no Curso de Ciência da Computação da Universidade do Extremo Sul Catarinense, UNESC, com Linha de Pesquisa em Engenharia de Software.

Criciúma, 28 de Junho de 2018.

BANCA EXAMINADORA

Profª. Ana Claudia Garcia Barbosa – MSc. - (UNESC) - Orientadora

Prof. Gilberto Vieira da Silva – Esp. - (UNESC)

Prof. Matheus Leandro Ferreira – Esp. - (UNESC)

A Deus que me deu forças para seguir em frente.

AGRADECIMENTOS

Agradeço à minha família pelo apoio e suporte para que eu pudesse concluir esta etapa fundamental em minha vida.

À pessoa que me acompanhou durante os momentos difíceis, e que compreendeu minhas inúmeras ausências.

Aos meus colegas e pessoas do meu cotidiano que me concederam motivação para continuar.

À minha orientadora Ana Claudia por me incentivar e me direcionar por todas as etapas desta pesquisa.

A Deus por ser a força dos meus dias.

“As pessoas apostam seus empregos, seu conforto, sua segurança, sua diversão, suas decisões e suas próprias vidas nos softwares de computadores. Eles precisam estar certos”

Roger S. Pressman

RESUMO

O mercado de aplicações móveis é exigente e espera que os sistemas atendam às suas necessidades sem imprevistos ou defeitos. As empresas de desenvolvimento de software com intuito de cumprir essas exigências utilizam o processo de teste de software como forma de garantir a qualidade do produto. O teste de software contribui com a construção de um produto sólido, confiável e que satisfaça a necessidade de seus usuários. Contudo, a implantação do processo de software é uma atividade desafiadora, pois exige a repetição contínua da execução dos testes. Este fator incentiva as empresas a realizar a utilização de estratégias de teste automatizado com objetivo de superar estes desafios. A especificação de casos de teste automatizado é um processo complexo e que exige conhecimento a respeito dos requisitos do sistema e das atividades que compõem o contexto de utilização dos usuários. Este cenário, em muitos casos, não demonstra a realidade das empresas onde há a presença de casos de teste com poucos detalhes e falta de rotinas exploratórias. Tendo em vista esse cenário, nesta pesquisa foi desenvolvido um procedimento que consiste em uma biblioteca de criação de testes automatizados de aspecto exploratório para aplicações móveis. Foram implementados algoritmos de registro de atividade e geração de casos de teste automatizado que possibilitaram a criação de casos de teste automatizado. Por meio da aplicação deste estudo realizou-se a criação de casos de teste automatizado com base no registro das atividades dos usuários de aplicações móveis.
Dentre os testes realizados e diante da análise dos casos de teste gerados confirmou-se uma melhoria na qualidade dos casos de teste automatizado com aspecto exploratório.

Palavras-chave: Engenharia de software. Aplicações Móveis. Automação de Testes. Testes Exploratórios.

ABSTRACT

The mobile application market is demanding and expects systems to meet the user needs without incidents or defects. Software development companies in order to meet these requirements have been using the software testing process as a way to ensure product quality. Software testing contributes to building a solid and reliable product that meets the needs of its users. However, deploying the software process is a challenging activity, as it requires continuous repetition of test execution. This factor encourages companies to carry out the use of automated test strategies in order to overcome these challenges. The specification of automated test cases is a complex process that requires knowledge about the system requirements and the activities that make up the usage context of users. This scenario, in many cases, does not describe the reality of companies where there is a presence of test cases with few details and lack of exploratory routines. Considering this scenario, in this research, a procedure has been developed that consists of a library for the creation of automated tests with an exploratory aspect for mobile applications. It has been implemented an algorithm for activity registration and generation of automated test cases that allowed the creation of automated test cases. Through the utilization of this study, the creation of automated test cases has been performed based on the records of the mobile user activities.
Among the performed tests and the analysis of the generated test cases, an improvement in the quality of the automated test cases with an exploratory aspect has been confirmed.

Keywords: Engeneering Software. Mobile Applications. Testing Automation. Exploratory Testing.

LISTA DE ILUSTRAÇÕES

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

API	*Application Programming Interface*
BDD	*Behavior Driven Development*
HTTP	*Hypertext Transfer Protocol*
REST	*Representational State Transfer*

SUMÁRIO

1 INTRODUÇÃO

O processo de testes nem sempre foi priorizado como uma atividade no processo de desenvolvimento. Em muitos casos, se o prazo para entrega do sistema não fosse o suficiente as empresas desconsideravam a execução das atividades do teste de software. Contudo, este comportamento vem se tornando cada vez menos frequente, dado ao fato de muitos defeitos serem encontrados em sistemas já liberados para os clientes, diminuindo a credibilidade do produto no mercado de trabalho, assim como a confiança dos clientes neste produto (SOMMERVILLE, 2011).

Por este motivo as empresas buscam na priorização do processo de testes uma forma amenizar este problema, além de ganhar a confiança e a credibilidade de seus clientes. As atividades exercidas durante o processo de testes consistem na detecção e correção de erros de software e não conformidades. Este processo busca validar o software diante das especificações garantindo que seus objetivos sejam atendidos, assim como as necessidades dos clientes (PRESSMAN; MAXIM, 2016).

O teste de software é uma atividade eficaz para a garantia de qualidade de software, porém a implementação é um grande desafio devido à alta complexidade dos sistemas e às inúmeras dificuldades relacionadas às etapas de desenvolvimento (BERNARDO, 2011).

Outro fator que torna o processo de testes uma atividade desafiadora é a necessidade de repetir os casos de teste durante o processo de desenvolvimento do software levando muitas empresas a optarem pela automação dos casos de teste (BASTOS et al., 2007).

A atividade de automação de testes pode ser definida pelo processo de reprodução dos passos de um usuário do sistema por meio da execução de casos de teste automatizado (MOLINARI, 2010).

Os casos de teste automatizados com características de teste exploratório são complexos e rico em detalhes, por este motivo possuem alto custo de criação e manutenção. Desta forma, costuma-se observar em empresas de desenvolvimento de software que muitos casos de teste não representam o contexto real da forma de utilização dos usuários. Diferente disto, nota-se casos de teste com poucas variações, e falta de detalhes, trazendo os vícios de implementação do

profissional responsável pela especificação dos testes (BARTIÉ, 2002).

Analogamente, os usuários exercem diferentes atividades durante a utilização do sistema. Estas atividades são variadas e ocorrem em diversos cenários que dificilmente serão previstos pelos responsáveis pela especificação dos testes automatizados devido às inúmeras possibilidades de caminhos a serem seguidos. Como consequência, muitos casos de teste não preveem comportamentos e ações inesperadas dos usuários (MORITZ, 2009, tradução nossa).

Quando não previstas pela equipe de qualidade, estas atividades permanecem sem a cobertura de testes exploratórios devida, expondo o sistema a cenários de vulnerabilidade em que a presença de defeitos pode ser revelada durante a sua utilização (PRESSMAN; MAXIM, 2016).

Sendo assim propõe-se por meio desta pesquisa um procedimento que consiste no desenvolvimento de uma biblioteca de criação de testes automatizados de aspecto exploratório para aplicações móveis.

Busca-se para minimizar o problema da falta de casos de teste exploratório por meio de uma melhoria na criação de testes automatizados utilizando o registro das atividades realizadas pelos usuários como forma de especificação dos casos de teste.

Serão consideradas as atividades de entrada e saída de dados, interação do usuário com a interface e navegação entre as áreas do sistema. Estas informações devem contribuir com o aumento das variações dos casos de teste e consequentemente com o aspecto exploratório dos testes, além de aumentar a fidelidade entre os passos reproduzidos pelo teste automatizado e as atividades dos usuários finais.

1.1 OBJETIVO GERAL

Propor um procedimento que possibilita a melhoria da criação de casos de teste automatizados com aspecto exploratório utilizando os registros das atividades e das interações dos usuários com as aplicações móveis.

1.2 OBJETIVOS ESPECÍFICOS

Os objetivos específicos desta pesquisa consistem em:

a) investigar algoritmos de teste automatizado para aplicações móveis;
b) implementar um algoritmo de registro de atividades;
c) implementar algoritmo de geração de casos de teste automatizado;
d) aplicar os algoritmos implementados para criação de casos de teste automatizado;
e) analisar os resultados dos casos de testes criados por meio desta pesquisa;

1.3 JUSTIFICATIVA

No processo tradicional de desenvolvimento de software deve-se submeter as aplicações móveis ao processo de teste para assegurar a qualidade do produto final. Se um defeito foi encontrado após disponibilização do produto o usuário poderá classificá-lo negativamente (MENEGASSI; ENDO, 2016, tradução nossa).

Segundo a pesquisa realizada pelo grupo *Standish Group* (2014, tradução nossa), mais de 61% dos projetos já perderam a data de entrega, mais de 59% perderam seus prazos devido ao orçamento. Ainda, destes, apenas 69% deles tiveram de suas funcionalidades validadas durante o ciclo de desenvolvimento com intuito de cumprir prazos. A decisão de colocar o produto antecipadamente no mercado pode ser crucial para a sobrevivência do produto, e em consequência disso a sobrevivência da empresa.

Como forma de garantia da qualidade do software em casos onde o processo tradicional de teste não pode ser aplicado, opta-se pela utilização de uma abordagem de testes exploratórios. A execução desta estratégia de testes, serve como uma forma de verificação final do software antes de determinar se ele está ou não pronto para ser liberado para o mercado. Utiliza-se também do teste exploratório em cenários onde não há um ponto de partida para a execução do processo de testes, desta forma, se faz necessário uma abordagem geral da situação do sistema de software (VARHOL, 2018, tradução nossa).

Devido aos motivos relatados acima é tão importante que hajam testes exploratórios que assegurem o funcionamento da aplicação das mais variadas maneiras, incluindo situações as quais o usuário raramente se depararia (MYERS; SANDLER; BADGETT, 2011, tradução nossa).

Neste tipo de teste, os testadores têm que fazer um esforço mínimo para o planejamento, mas a cobertura da execução dos testes deve ser máxima para que o testador obtenha a funcionalidade exata da aplicação. Isso pode ser útil para o testador decidir o que pode ser considerado um teste válido para o produto. Ao testar, o testador aprende sobre o comportamento do aplicativo de software, dando início a criação de um caso de teste ou cenários de teste (SILVA; ALVES; BRUNO, 2011).

Entende-se a importância de casos de teste exploratórios para garantia da qualidade do sistema de software e da diversificação dos passos especificados nos casos de teste, visto que isto aumenta as chances da detecção de erros no sistema. Contudo, na maioria dos cenários não é isso que ocorre. O que se observa, ao invés disso, são testes que possuem poucas variações, e pobres em detalhes costumam não prever comportamentos/ações inesperadas dos usuários (PRESSMAN; MAXIM, 2016).

Ainda, a qualidade das aplicações móveis é insuficiente, e na maioria das vezes devido ao rápido processo de desenvolvimento adotado, no qual as atividades de teste são negligenciadas ou aplicadas de forma superficial por serem consideradas complexas, difíceis de automatizar, caras e/ou demoradas (AMALFITANO; FASOLINO; TRAMONTANA, 2011, tradução nossa).

Sabe-se que os usuários de aplicações móveis executam as mesmas tarefas de maneiras distintas. A forma de utilização é variável em vários aspectos, incluindo a navegação na aplicação ou mesmo os dados inseridos. Estes fatores compõem a diversidade do contexto de utilização de cada usuário.

É com base nesta diversidade que o trabalho corrente foi baseado. Acredita-se que este procedimento seja uma abordagem eficaz para melhoria da cobertura de testes exploratórios, pois propõem a utilização do registro das atividades dos usuários de aplicações móveis para realizar a especificação dos casos de teste automatizado.

A implementação desta pesquisa mostra-se relevante no auxílio da criação de casos de teste com aspecto exploratório em aplicações móveis

contribuindo para que que se obtenha o aumento da cobertura de casos de teste com riqueza em detalhes e variações.

1.4 ESTRUTURA DO TRABALHO

Esta pesquisa aborda diferentes aspectos do processo de garantia de qualidade de software divididos em seis capítulos.

No capítulo da Introdução no qual o corrente texto se encontra, trata-se de apresentar o trabalho proposto expondo. Realizou-se o esclarecimento da motivação que levou a confecção deste trabalho. Determinou-se os objetivos gerais e específicos com propósito de resolver o problema da falta de casos de teste exploratório e finalizando com a justificativa para realização do trabalho.

No decorrer do capítulo sobre Teste de Software descreve-se as etapas do ciclo de vida do teste de software, apontando os modelos de processo mais utilizados. São abordadas as etapas do processo de teste, e as fases que determinam a execução de diferentes tipos de teste. Busca se também, identificar as especificidades relacionadas ao teste de software para aplicações móveis.

O conteúdo abordado no capítulo de Automação de testes abrange o processo de automação de testes de forma detalhada, mediante a um texto que elabora as dificuldades da implantação da automação e as possíveis soluções a serem adotadas. Também foram abordados assuntos envolvendo a utilização de ferramentas de automação de testes.

O referencial teórico consultado no capítulo de Trabalhos Correlatos relaciona os assuntos abordados nesta pesquisa à trabalhos acadêmicos semelhantes utilizados como base de conhecimento.

Elucida-se no quinto capítulo a metodologia utilizada para aplicação deste trabalho, abordando as etapas realizadas para a efetivação dos objetivos propostos e a obtenção dos resultados. São esclarecidas as etapas de desenvolvimento dos algoritmos de registro de atividades e de geração de casos de teste automatizado e aplicação destes algoritmos em uma aplicação móvel de testes. Por último, aborda-se os resultados obtidos por meio da execução dos algoritmos e dos testes realizados durante a etapa de implementação da pesquisa.

No capítulo de Conclusão, é exercida uma reflexão sobre as contribuições desta pesquisa, dificuldades enfrentadas e as possíveis pesquisas futuras seguindo linha de estudo semelhante a esta.

2 TESTE DE SOFTWARE

O teste de software é um processo projetado para garantir que o software esteja de acordo com o que foi definido em seus requisitos tornando o software previsível e consistente, sem erros ou anomalias (MYERS; SANDLER; BADGETT, 2011, tradução nossa).

Quando conduzido de forma correta o processo de teste de software tende a demonstrar erros no software, e desta forma, validar se as funcionalidades do software estão de acordo com suas especificações funcionais, além de garantir seu funcionamento. O teste busca determinar se existem erros nestas funcionalidades, porém não é possível afirmar que o software esteja totalmente livre de erros (PRESSMAN; MAXIM, 2016).

O processo de teste envolve atividades de verificação e validação que garantem os requisitos funcionais do software. A atividade de verificação busca garantir que o produto está sendo construído da maneira correta. O objetivo da atividade de validação do software é garantir que as funcionalidades do software atendam às expectativas do cliente. Os processos de verificação e validação permitem que se possa dizer se um software está devidamente implementado para atender o seu objetivo (SOMMERVILLE, 2011).

Antes de iniciar a implantação de testes de software é necessário que se considere que alguns investimentos serão necessários para preparar as equipes, providenciar equipamentos e instalações. Isto pode gerar algumas incertezas sobre os ganhos que o teste pode proporcionar. Contudo, a implementação do teste de software traz benefícios que impactam positivamente o processo de garantia de qualidade de software aumentando a confiabilidade no que está sendo produzido. Os resultados são o aumento da lucratividade e da produtividade além de uma melhoria na relação com os clientes devido a entrega de produtos com garantia de qualidade (BASTOS et al, 2007).

A implantação do teste de software refina o processo de produção e manutenção do sistema. Isto ocorre em consequência das atividades de verificação e validação do sistema que realizam a conferência do sistema desde a sua documentação, envolvendo manuais, requisitos funcionais e do próprio software. Como resultado deste processo a documentação do produto deverá estar de acordo com os requisitos definidos assim como o produto final (RIOS; MOREIRA, 2006).

Este refinamento do processo de produção e manutenção do sistema provoca uma redução no tempo e custo de trabalho com pedidos de ajustes e correções. Estas correções quando feitas durante o processo de modelagem ou definição do sistema possuem um menor custo quando comparadas a correções feitas em ambiente de produção (MYERS; SANDLER; BADGETT, 2011, tradução nossa).

Entende-se que o processo de teste visa garantir que os erros no software sejam encontrados e resolvidos durante o a modelagem e definição do sistema, ou seja, antes dos clientes. Além disso procura-se determinar se software está de acordo com o que foi definido em sua especificação. Apesar da eficácia deste processo na obtenção da qualidade, o teste de software requer que sua implementação seja realizada da forma planejada para garantir que o produto possua o nível de qualidade e confiabilidade esperado (RIOS; MOREIRA, 2006).

Para isto é importante que se o compreenda um pouco sobre o ciclo de vida do teste de software e as etapas nas quais o teste pode ser dividido. O subcapítulo seguinte abrange com mais detalhes as características de cada etapa do ciclo de vida do teste de software.

2.1 O CICLO DE VIDA DO TESTE DE SOFTWARE

O ciclo de testes consiste na aplicação de uma abordagem formada por técnicas e rotinas que auxiliam na organização da execução dos testes de software de forma estruturada garantindo que todas as etapas do processo sejam concluídas de forma correta (RIOS; MOREIRA, 2006).

2.1.1 Modelos de processo

Existem diversos modelos de processo de teste e desenvolvimento de software. Os modelos mais conhecidos são (PRESSMAN; MAXIM, 2016; SOMMERVILLE, 2011):

a) **modelo espiral**: a execução das atividades ocorre de forma sequencial e contínua, onde há a ligação da etapa atual com a próxima etapa. Estas etapas são divididas em regiões de acordo com as características especificas das atividades envolvidas. A atividade

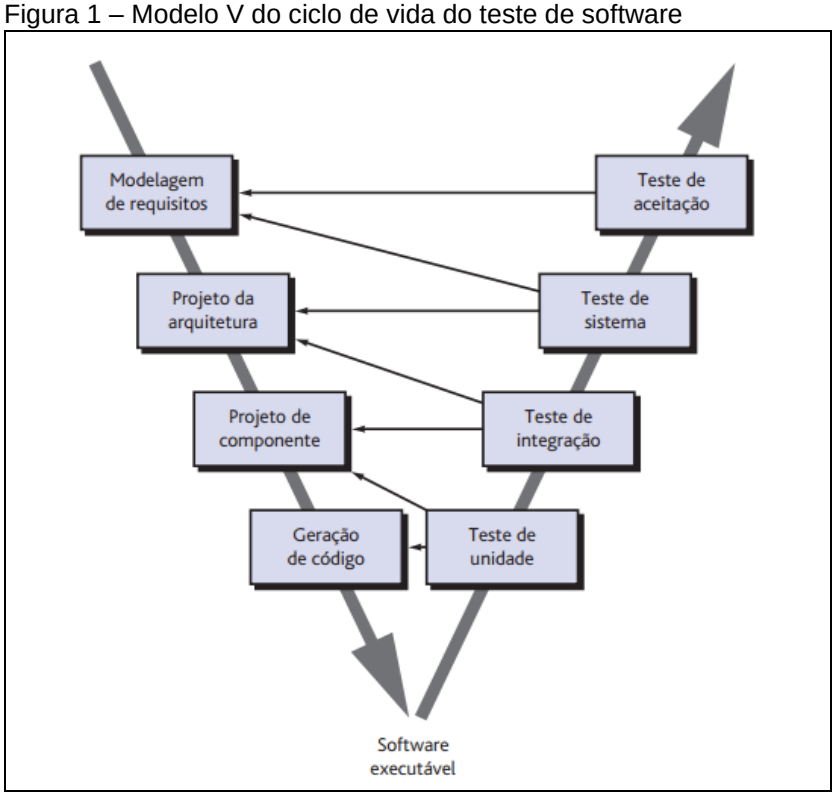
de teste e de desenvolvimento do software são agrupadas em uma mesma região;

b) **modelo em cascata:** é formado pela combinação de técnicas de etapas sequenciais lineares para construir a abordagem de incremento em camadas. As etapas são iniciadas a partir da finalização da etapa anterior. Neste processo as tarefas podem ser divididas de acordo com cada etapa. A saída de uma etapa torna-se entrada da próxima. Este processo permite que as etapas anteriores sejam revisadas e durante as etapas do próximo ciclo;

c) **modelo V**: pode ser considerado uma extensão do modelo em cascata. Seu objetivo consiste em melhorar a eficiência e eficácia de atividades de desenvolvimento de software e refletir relação entre as atividades de teste e desenvolvimento.

Entre os modelos de processo abordados, o modelo V apresenta uma maior fluidez no processo de testes. Este modelo busca assegurar que o processo de garantia de qualidade e em conjunto com os testes ocorram ao longo do ciclo de vida do processo de desenvolvimento do sistema.

Figura 1 – Modelo V do ciclo de vida do teste de software



Fonte: Pressman e Maxim (2016).

Em sua representação o modelo V é composto por dois lados. Do lado esquerdo se encontram as atividades de desenvolvimento e atividades de preparação de teste correspondentes. Enquanto do lado direito estão localizadas as atividades de execução e depuração do teste, como ilustrado na figura 1. As setas entre a implementação do sistema e as execuções dos testes descrevem as iterações que envolvem verificações e validações do sistema (ABDULRAZEG; NORWAWI; BASIR, 2014, tradução nossa).

Conclui-se que os modelos de teste abordados abrangem as principais características de cada processo, e, portanto, fornecem apenas informações parciais sobre estes processos. Por este motivo é importante que haja um estudo sobre a compatibilidade da implantação de cada modelo de processo de acordo com as necessidades de teste de cada projeto.

2.1.2 Etapas do processo de teste

É importante salientar que além dos modelos de processo, o ciclo de vida do teste é composto por etapas que auxiliam na organização e execução da atividade de teste. Estas etapas podem ser definidas por (RIOS; MOREIRA, 2006):

a) **procedimentos iniciais:** possui a finalidade de definir os requisitos do sistema a serem testados levando em consideração a prioridade de cada teste. Nesta etapa são estabelecidas as estratégias de teste que serão utilizadas;

b) **planejamento:** ocorre a elaboração da estratégia de teste e o caso de teste a serem utilizados. Realiza-se a estimativa do tempo que se deseja investir neste processo, os recursos que serão alocados para a atividade de teste, assim como as responsabilidades que serão atribuídas. Além disso, os resultados esperados destes testes são documentados;

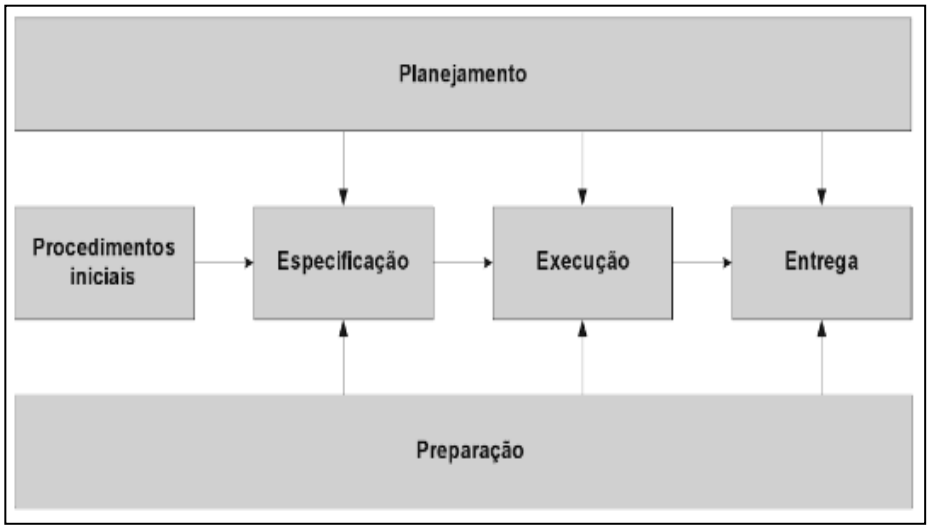
c) **preparação**: durante esta etapa busca-se garantir que o ambiente de testes esteja preparado para a execução dos testes, incluindo equipamentos, pessoal, ferramentas de automação, hardware e software. Esta etapa ocorre em paralelo com as outras etapas;

d) **especificação**: nesta etapa ocorre a elaboração dos roteiros e dos casos de teste que serão executados. Os testes devem ser documentados;

e) **execução:** o software é submetido a rotina de testes utilizando os casos de teste especificados. Se houverem casos de teste para serem executados de maneira automatizada será necessária a utilização de ferramentas de automação;

f) **entrega**: ocorre a documentação dos resultados dos testes, avaliação dos resultados da execução dos testes para que se possa realizar a validação do software, onde o responsável pela definição dos requisitos poderá avaliar se o que está sendo implementado é compatível com o que foi especificado considerando os resultados obtidos.

Figura 2 – Modelo 3P x 3E do ciclo de vida do teste de software



Fonte: Rios e Moreira (2006).

O modelo do processo de teste 3P x 3E encontra-se ilustrado na figura 2. Considerando a capacidade de organização e gerenciamento exposta por este modelo, optou-se por realizar a utilização desta abordagem para aplicação do estudo aqui presente.

Entende-se por meio do texto abordado que a implantação de um modelo de teste compatível somado a execução bem-sucedida das etapas do processo de teste tornam o processo de teste eficaz e confiável. Isto faz com que o processo estruturação dos testes se torne menos custoso.

2.2 FASES DE TESTE

As atividades de teste podem ser divididas em fases de teste acordo com a complexidade e o tamanho de cada projeto.

Os objetivos de cada fase de teste são específicos e podem ser aplicados em diferentes partes do sistema. As fases de teste de software são: teste de unidade, teste de integração, teste de sistemas e teste de aceitação.

Estas fases possibilitam estabelecer um nível de testabilidade e qualidade do software. À medida que ocorre a execução de cada uma das fases, verifica-se a maturidade do sistema e prossegue-se para uma próxima fase, definindo assim se o sistema está pronto para ser liberado para o mercado.

2.2.1 Teste de Unidade

O teste de unidade, também conhecido como teste unitário, é a fase do processo de teste que verifica os menores componentes do sistema testando a lógica interna do software dentro de um escopo isolado para garantir sua funcionalidade (PRESSMAN; MAXIM, 2016).

Nesta fase de teste ocorre a verificação de toda a estrutura interna do software validando o maior número possível de caminhos e fluxos condicionais existentes do sistema. Preocupa-se com os menores componentes do software com o intuito de garantir que as informações que entram e saem do sistema são consistentes, mesmo quando em condições limites. Busca-se garantir que o sistema mantenha sua execução de forma correta por todos os caminhos, além de garantir que as informações ou condições que provocam erros no sistema estejam devidamente tratadas (MYERS; SANDLER; BADGETT, 2011, tradução nossa).

Com a prática do teste de unidade a depuração de defeitos se torna menos custosa visto que o teste é focado em trechos internos do código, desta forma, quando um dos testes falha, é possível saber exatamente o ponto do código em que a falha ocorreu (BERNARDO; KON, 2008).

Entende-se então que o teste de unidade é uma prática indispensável para o processo de garantia de qualidade de software, visto que sua implementação permite detectar de forma precisa quando houverem possíveis falhas ou alterações

indesejadas introduzidas no sistema, evitando maiores custos com ajustes e correções.

2.2.2 Teste de Integração

Durante o teste de integração ocorre o agrupamento dos componentes que haviam sido testados isoladamente, formando subsistemas de acordo com a arquitetura definida para o sistema (SOMMERVILLE, 2011).

Busca-se com este teste garantir que todas as partes do sistema estejam se comunicando de forma correta validando a arquitetura do software, ao mesmo tempo em que se realizam testes para descobrir erros associados às interfaces (PRESSMAN; MAXIM, 2016).

A execução dos testes de integração pode ser realizada de duas formas: incremental e não incremental.

A forma não-incremental, também conhecida como *big-bang*, visa o agrupamento de todos os componentes durante o processo de teste de integração. Isto pode levar a resultados não satisfatórios, visto que realizar a rastreabilidade e o isolamento dos erros encontrados durante os testes se torna custoso dado a extensão do programa (MYERS; SANDLER; BADGETT, 2011, tradução nossa).

Por outro lado, a integração incremental aborda uma metodologia que visa realizar a execução dos testes em paralelo com a construção do sistema. Acompanhando cada vez que um componente é incrementado ao sistema. Isto permite isolar os erros de forma mais fácil, além de garantir uma maior probabilidade de que as interfaces sejam testadas completamente (PRESSMAN; MAXIM, 2016).

Algumas estratégias de teste são utilizadas em conjunto com a integração incremental. Estas estratégias podem ser definidas por (NAIK; TRIPATHY, 2008, tradução nossa):

a) ***bottom-up***: durante a fase inicial do teste os componentes e os módulos do mais baixo nível na estrutura do programa são agrupados iniciando formando novos módulos ou subsistemas até que haja a integração com componentes de níveis superiores. Isto permite que haja uma verificação antecipada dos comportamentos de baixo nível do sistema;

b) ***top-down***: segue uma abordagem inversa a estratégia *bottom-up*,

> iniciando o agrupamento dos componentes, módulos e subsistemas a partir dos níveis superiores, incorporando os módulos principais do programa. Esta estratégia permite que haja a verificação e a validação antecipada dos comportamentos de alto nível. Além disso, é possível realizar demonstrações de resultados aos usuários antes mesmo que os níveis inferiores do sistema estejam completamente implementados e construídos. Isto requer que haja a criação de artifícios para simular a ausência destes níveis.

Conclui-se por meio do texto abordado que a fase de teste de integração possibilita determinar a maturidade do sistema ao se integrar com outras áreas. Pode-se entender por meio desta abordagem se aplicação está pronta para uma abordagem de testes de alto nível, onde há integração dos componentes com as outras partes do sistema.

2.2.3 Teste de Aceitação

O teste de aceitação permite a um restrito grupo de usuários validar a conformidade do sistema desenvolvido com seus requisitos (LEUNG; YEUNG, 2007, tradução nossa).

As principais características do software são validadas com intuito de garantir que o seu funcionamento e o aspecto do software sejam aceitos pelos usuários. O software será avaliado com base nas necessidades funcionais que devem ser atendidas e os objetivos definidos devem estar implementados de acordo com as especificações (BOUCHER; MUSSBACHER, 2017, tradução nossa).

Em alguns casos os usuários podem definir critérios de aceitação para decidir se o software possui o mínimo para atender aos seus requisitos. Verifica-se se as entradas e as saídas de dados no sistema após determinados processos continuam de acordo com o que o especificado nos critérios de aceitação (WEISS; SCHILL, 2016, tradução nossa).

2.2.3 Teste de Sistema

O teste de sistema busca determinar se as características funcionais do software estão de acordo com o que foi definido em sua especificação por meio de

um processo de verificação e validação (MYERS; SANDLER; BADGETT, 2011, tradução nossa).

A execução desta fase ocorre após a integração dos módulos do sistema incorporando o software a outros elementos, como hardware, e informações externas (PRESSMAN; MAXIM, 2016).

O objetivo desta fase é encontrar falhas no software por meio da execução de rotinas de teste que se assemelhem ao comportamento do usuário final. Preocupa-se em reproduzir um ambiente com as mesmas características do ambiente do usuário final. A execução dos testes é realizada utilizando dados de entrada que um usuário utilizaria (ROCHA; MALDONADO; WEBER, 2001).

Entende-se pelo texto abordado que as fases de teste possuem diferentes abordagens e objetivos. Desta forma, é importante que haja uma atividade com intuito de definir o que deve ser testado, focando em pontos críticos, e desta forma decidir entre as diferentes técnicas de teste existentes quais suprem as necessidades para garantir a qualidade e confiabilidade dos requisitos do produto (RIOS; MOREIRA, 2006).

A fase de teste de sistema se mostrou compatível com os objetivos propostos por esta pesquisa, pois indica a realização dos testes de software em uma fase posterior de desenvolvimento. Busca-se com a utilização desta abordagem prover uma abordagem de alto nível dos testes, onde há um histórico de execuções das fazes anteriores garantindo a maturidade de testabilidade do sistema.

Conclui-se sobre essa subseção que a compreensão sobre as fases de teste que compõem o processo de software é fundamental, e deve ser aliada ao conhecimento das técnicas de teste utilizadas para garantir a conformidade dos sistemas de software.

2.3 TÉCNICAS DE TESTE

As diferentes técnicas de testes que formam o processo de teste são chamadas de teste caixa branca e teste caixa preta. Estes, possuem diferentes características específicas em relação a suas áreas de abrangência, forma de implementação e custos (PRESSMAN; MAXIM, 2016).

2.3.1 Teste Caixa Branca

A técnica de teste denominada de teste caixa branca, também conhecida como teste estrutural ou teste movido a lógica, permite verificar a estrutura interna do software e seu funcionamento após exposição de valores de entrada para todos os fluxos do programa buscando encontrar falhas (MYERS; SANDLER; BADGETT, 2011, tradução nossa).

Esta técnica preocupa-se em detectar se os comandos utilizados no sistema estejam funcionando corretamente, verificando a estrutura interna em busca de falhas em sua implementação, variáveis utilizadas que não estejam definidas ou que a inicialização e finalização dos laços de repetição não tenham sido implementados corretamente (SOMMERVILLE, 2011).

2.3.2 Teste Caixa Preta

No teste caixa preta, também conhecido como teste funcional, o comportamento interno e a estrutura do programa não são considerados. O testador que utiliza o método de teste caixa preta não precisa saber sobre a lógica interna ou estrutura do programa. Ao invés disso, o teste de caixa preta se concentra em encontrar situações em que o seu comportamento difere do que foi definido em seus requisitos (MALL, 2009, tradução nossa).

Por meio do teste caixa preta é possível realizar a validação do software por meio de diversos tipos de testes funcionais buscando encontrar defeitos e não conformidades no sistema, entre eles podem ser citados os testes de (BASTOS et al., 2007):

a) requisitos: visam verificar se as funcionalidades implementadas no software estão de acordo com os requisitos dos usuários. Busca-se isolar defeitos nas funcionalidades comparando com o esperado do produto final. A execução dos testes é feita em forma de checklist garantindo que cada uma das implementações esteja de acordo com o que foi definido em um processo que evolui à medida que o sistema se torna mais completo e com mais implementações;

b) regressão: verifica o software em busca de falhas e defeitos que possam ter sido causados por alterações em diferentes partes do

sistema. Busca-se garantir que as funcionalidades previamente testadas continuem funcionando;

c) **tratamento de erros**: verifica a capacidade do sistema de se recuperar após operações incorretas. Busca-se determinar se o sistema reconhece todas as condições de erro esperadas e se para cada um destes erros o sistema é capaz de tratar estas informações sem prejudicar o fluxo e a utilização do software;
d) **suporte manual**: verifica se a documentação e o procedimento de suporte manual estão complementos. Visa validar se as responsabilidades pelo suporte manual estão definidas. Os testes devem ocorrer no início do ciclo de vida do software e sendo mantidos em paralelo com as etapas de entrega e implantação do software;
e) **interconexões**: visa determinar se os dados e os parâmetros inseridos no sistema estão sendo transferidos corretamente entre os softwares garantindo a interconexão entre eles. Busca certificar que o momento da execução entre as tarefas dos softwares esteja correto e que haja a coordenação das funções entre os softwares;
f) **controle**: assegura que os dados do software estejam inseridos de forma íntegra, garante que a realização de transações no software seja autorizada e que os dados recebidos sejam auditáveis. Também avalia se o processamento das informações é realizado de forma eficiente, eficaz e econômico atendendo as necessidades funcionais;
g) **paralelos**: determina se os resultados dos testes de uma nova versão do software são consistentes com as versões anteriores, assegurando que a nova versão execute corretamente. Serão considerados testes com resultados satisfatórios aqueles que não encontrarem defeitos após a análise comparativa dos resultados dos testes realizados nas funcionalidades.

O teste de regressão se mostrou uma solução viável para o aprimoramento do processo de teste de software ampliando a qualidade do produto, diminuindo custos e prazos durante a produção do software. A execução deste tipo de teste pode vir a revelar defeitos em áreas do software que não tenham sido alteradas e que devem prosseguir funcionando da mesma forma como anterior as alterações (PRESSMAN; MAXIM, 2016).

Conclui-se que conhecer sobre os tipos de testes existentes e seu funcionamento é importante para o processo de testes, pois permite que se possa determinar quais tipos de testes são adequados a estrutura de desenvolvimento e teste de cada sistema. Além disso, é possível aprimorar ainda mais o processo de garantia de qualidade de software através da implantação de estratégias de testes com caráter exploratório.

2.3.2.1 Teste Exploratório

O teste exploratório consiste em uma estratégia de execução da atividade de testes em paralelo com o processo de aprendizagem, documentação e especificação de novos casos de teste (PFAHL et al., 2014, tradução nossa).

A abordagem de testes exploratórios permite a execução de atividades de validação em casos onde não há a presença de um roteiro de testes pré-definido. A execução destas atividades costuma ser realizada com base na experiência do testador, ou de um especialista, buscando encontrar inconsistências no sistema que não tenham sido previstas durante a etapa de especificação, ou em casos onde os casos de teste encontram-se obsoletos (GEBIZLI; SOZER, 2017, tradução nossa).

A execução da atividade de testes exploratórios contribui com o processo de testes de forma positiva, pois valida o software de forma horizontal, garantindo que as funcionalidades de diferentes módulos e cenários do sistema estejam de acordo com a sua especificação. Estas contribuições podem ser definidas da seguinte maneira (RAAPPANA et al., 2016, tradução nossa):

a) **previne repetições:** evita que a especificação dos casos de teste contemplem cenários de forma repetitiva, já abordados por diferentes casos de teste ou testadores;
b) **aumento da cobertura**: são contempladas partes do sistema que não tenham sido consideradas na especificação dos casos de teste, além disso, utiliza-se uma abordagem de testes mais abrangente tendo em mente as rotinas de teste que podem vir a revelar defeitos no sistema;
c) **detecção de não conformidades:** auxilia no aprimoramento da detecção de não conformidades no sistema, pois contempla maior diversidade de cenários de teste;
d) **aperfeiçoamento da documentação**: permite um visão mais acurada

do processo de testes, facilitando na tomada de decisões gerenciais.

A abordagem exposta pelo teste exploratório se mostrou compatível com a aplicação deste estudo, pois possibilita uma maior cobertura da garantia da qualidade do sistema assemelhando o processo de testes com a forma de utilização realizada pelo usuário abrangendo além das rotinas principais, onde é esperado que tudo esteja funcionando. Isto é, busca-se encontrar não conformidades que podem ter sido desconsideradas durante a etapa de especificação dos casos de teste.

Além da importância de entender sobre as fases, tipos e técnicas de teste envolvidas no processo de garantia de qualidade de software, é válido ressaltar a importância do conhecimento sobre as diferentes estratégias utilizadas para a abordagem de testes utilizadas em dispositivos móveis, que possuem grande relevância para esta pesquisa (GAO et al., 2014, tradução nossa).

2.4 TESTES PARA DISPOSITIVOS MÓVEIS

A atividade de teste para dispositivos moveis possui o objetivo de determinar se o a implementação do aplicativo atende aos requisitos definidos, além de garantir que sua funcionalidade esteja correta considerando aspectos específicos de usabilidade, navegabilidade, desempenho, capacidade e segurança do aplicativo (PRESSMAN; MAXIM, 2016).

Garantir a qualidade de aplicações móveis por meio do processo de testes tem se tornado cada vez mais comum entre as empresas de desenvolvimento de software. Esta prática vem se tornando comum devido ao aumento do crescimento da tecnologia móvel que tem se mostrado significativamente rápido em comparação ao aumento do crescimento de outras tecnologias. Este crescimento vem atraindo as empresas que tendem a diminuírem os prazos de entrega de seus produtos para se destacar no mercado. Desta forma, busca-se no processo de testes uma forma de atender a demanda do mercado dentro dos prazos exigidos sem perder a qualidade dos produtos desenvolvidos (SOASTA, 2011, tradução nossa).

Contudo, a adoção da prática de testes para aplicações móveis ainda é um desafio devido aos vários requisitos únicos que distinguem os testes de aplicativos móveis dos testes convencionais de software. Estes requisitos estão

presentes em vários aspectos da atividade de testes e a sua prática está diretamente relacionada à uma implantação bem-sucedida do processo de testes.

Os requisitos que devem ser considerados para a execução de testes para dispositivos moveis podem ser definidos da seguinte maneira (GAO et al., 2014, tradução nossa):

a) **modelos de dispositivos**: a grande diversidade de modelos de dispositivos móveis e as restrições técnicas e comerciais relacionadas a estes dispositivos;
b) **capacidade dos dispositivos**: os aspectos das diferentes limitações dos dispositivos, incluindo memória, processamento, capacidade de armazenamento e velocidade de conexão com as redes sem fio;
c) **recursos de energia**: a durabilidade da bateria dos dispositivos e o gerenciamento da energia;
d) **tamanho de visores**: a responsividade da interface do aplicativo em diferentes tamanhos de *displays*, densidade de visores, e resoluções;
e) **protocolos de rede**: os protocolos e tecnologias de rede utilizadas pelos diferentes modelos de aparelhos móveis;
f) **plataformas**: a diversidade das plataformas e sistemas operacionais existentes suportados pelo aplicativo;
g) **aprovação da publicação nas lojas de aplicativos**: as exigências que devem ser atendidas pelo produto para que possa ser publicado em lojas de aplicativos, incluindo diretrizes para validação de interface e padrões de comportamento pré-definidos;
h) **necessidades específicas de usuários**: requisitos específicos que envolvem diferentes necessidades associadas a utilização de aplicativos móveis e necessidade de ajustes de recursos disponíveis em dispositivos móveis;
i) **prazos**: o tempo disponível para execução de testes e validações é menor devido à grande demanda por novas versões para atender aos requisitos do mercado;
j) **simulação de dispositivos:** para validação e verificação da utilização do aplicativo em diferentes dispositivos diminuindo a necessidade de possuir os dispositivos reais;

k) **conectividade**: diferentes cenários de estado de conexão, níveis de sinal de rede e disponibilidade de conexão com a internet.

Para elaboração desta pesquisa serão considerados os requisitos citados buscando aproximar a implementação e aplicação desta pesquisa ao cenário de testes das empresas de aplicativos móveis.

Diante dos diferentes requisitos e fatores que influenciam a execução das atividades de teste em dispositivos móveis é necessário que se considere uma abordagem de testes diferenciada que contribua com as estratégias de teste convencionais existentes. Estas estratégias visam garantir a funcionalidade correta dos aplicativos móveis diante dos desafios impostos pelo cenário de testes em dispositivos móveis. As estratégias de teste para dispositivos móveis podem ser definidas por (GAO et al., 2014, tradução nossa):

a) **teste de funcionalidade e comportamento**: busca garantir o funcionamento do dispositivo validando seu comportamento mediante as interações do usuário, interações com ambientes externos, funcionalidade das APIs e a independência do sistema;

b) **teste de qualidade de serviço:** avalia o desempenho do sistema mediante ao carregamento de informações, a disponibilidade do sistema, a capacidade de crescimento e escalabilidade e vazão de informações;

c) **teste de interoperabilidade:** verifica a capacidade do sistema de se comunicar e interagir com diferentes dispositivos, plataformas, *browsers* e redes sem fio;

d) **teste de usabilidade e internacionalização:** preocupa-se em garantir o funcionamento da interface, alertas exibidos aos usuários, a disponibilidade e funcionalidade dos diferentes fluxos de navegação, as informações de mídia e conteúdo visual e o suporte à interação com gestos;

e) **teste de segurança e privacidade:** atividade que visa garantir a autenticação dos usuários, segurança do dispositivo e das informações armazenadas, segurança das sessões criadas em transação externas, segurança da comunicação entre redes, segurança de comunicação e a privacidade do usuário;

f) **teste de mobilidade:** valida o a conformidade de funcionalidades do dispositivo baseadas na localização do usuário, perfis de utilização, dados do sistema e informações dos usuários;

g) **teste de compatibilidade e conectividade:** verifica a capacidade de funcionamento dos recursos de rede em diferentes navegadores, plataformas e redes sem fio;

h) **teste de multi-tenacidade:** verifica as informações, interface, e comportamento do sistema diante das funcionalidades utilizadas por múltiplos usuários a partir de uma única instância do software.

As estratégias de teste citadas acomodam a atividade de testes em cenários de testes em dispositivos móveis quando executadas em conjunto com as estratégias convencionais de teste já tratadas em subcapítulos anteriores. Contudo, estas estratégias não diminuem o fato da dificuldade da implantação e execução do processo de testes em aplicações móveis (KNOTT, 2015, tradução nossa).

Conclui-se que a dificuldade da implementação de testes para dispositivos móveis fica ainda mais clara em cenários onde o produto é constantemente submetido a alterações e correções. Nestes casos, pode se tornar inviável manter uma abordagem de testes de forma manual para garantir a integridade do sistema. Desta forma, uma abordagem eficaz para amenizar este problema é a implantação de testes automatizados para garantir a qualidade e confiabilidade do sistema. O capítulo seguinte realiza uma abordagem mais detalhada sobre o processo de automação de testes.

3 AUTOMAÇÃO DE TESTES

A automação de testes possui o objetivo de exercitar o sistema por meio da execução de atividades de teste de forma automatizada verificando os requisitos e validando as funcionalidades do sistema (BERNARDO, 2008).

Esta atividade vem ganhando destaque com o aumento da preocupação das grandes empresas em garantir a qualidade de seus produtos reduzindo prazos e custos. Contudo, é importante salientar que a implantação do processo de automação de testes exige determinada maturidade da equipe responsável pela atividade de testes, sendo necessário o conhecimento dos requisitos e especificações do sistema, e a compreensão das técnicas e ferramentas utilizadas no processo de automação de testes (MOLINARI, 2010).

Ainda, é necessário enfatizar a necessidade da validação dos resultados dos casos teste automatizados, distinguindo os erros do software das falhas de especificação dos casos de teste. Esta atividade deve ser realizada por profissionais que possuam maturidade na execução de tarefas de teste manual e conhecimento das regras de negócio do sistema (RIOS; MOREIRA, 2006)

De forma geral, a implantação da automação de testes resulta no aumento da qualidade e confiabilidade dos testes, visto que o processo de automação possibilita a reprodução das sequencias de teste especificas por meio de casos de teste e ferramentas de automação (BERNARDO, 2008).

A execução da atividade de automação pode ser repetida inúmeras vezes por longos períodos de tempo, isto faz com que o comportamento do sistema possa ser validado com mais precisão, e que os resultados obtidos sejam mais assertivos (WISSINK; AMARO, 2006, tradução nossa).

A sistematização da atividade de testes manuais por meio da automação diminui as chances de erros em rotinas de execução repetitiva onde o resultado dos testes pode diferir do resultado esperado (MOLINARI, 2010).

Estes erros podem ocorrer devido a vários fatores, incluindo a experiência do testador, qualidade do caso de teste, interpretação dos requisitos de teste e baterias de teste extensas que podem levar a exaustão dos testadores e consequentemente a perda do foco na atividade de testes (BARTIÉ, 2002).

O processo de automação pode ser utilizado como parte da solução para este problema, por meio da execução assertiva de casos de teste que buscam

determinar a qualidade e confiabilidade do sistema de software. (SOMMERVILLE, 2011).

Além disso, as razões pelas quais a equipe de testes deve optar pela implantação do processo de automação de testes podem ser definas da seguinte maneira (CRISPIN; GREGORY, 2008, tradução nossa):

a) o tempo necessário para execução de testes automatizados é menor que o tempo do teste manual;
b) os erros cometidos durante as atividades de teste são menos frequentes, visto que o processo ocorre de forma automatizada e sistemática;
c) existe uma diminuição do tempo necessário para execução das atividades de teste automatizado resultando no ganho de tempo para aperfeiçoamento do processo;
d) as alterações no sistema podem ser realizadas com mais segurança e apoiando-se na execução dos testes automatizados;
e) o feedback provido pelo resultado dos testes automatizados ocorre com mais rapidez e de forma frequente;
f) os testes automatizados permitem direcionar a codificação por meio da utilização de técnicas de TDD, além de garantir a cobertura dos testes de regressão;
g) a atividade de testes automatizados prove documentação de qualidade;
h) os gastos com a atividade de automação são recuperados ao longo do tempo por meio da economia com gastos em correções.

Entre os benefícios citados, pode-se citar o curto tempo de resposta e entrega de resultados por meio do *feeback* de testes como fator de grande importância, visto que isso permite a equipe de testes determinar se as alterações realizadas introduziram novos defeitos no sistema.

Como parte do processo de implantação da automação de testes, devem ser considerados alguns fatores que podem vir a desencorajar algumas empresas a optarem pela atividade de automação de testes, assim como as dificuldades enfrentadas durante a fase de implantação. No subcapítulo a seguir serão abordados com um maior detalhamento os fatores envolvidos na implantação da atividade de automação de testes.

3.1 IMPLANTAÇÃO DA AUTOMAÇÃO DE TESTES

Inicialmente, algumas empresas se deparam com um elevado custo para realizar a implantação do processo de testes automatizados, envolvendo à a compra de licenças para ferramentas de automação, treinamento e capacitação da equipe responsável pelos testes e contratações de pessoal qualificado. Estes gastos muitas vezes levam as empresas a desistirem da automação de testes por acreditarem que o retorno desta atividade será menor do que o investimento. Contudo, deve-se considerar que os gastos iniciais são necessários, devido a necessidade de criação dos casos de teste e treinamentos para a equipe de testes, que mais tarde será convertido em retorno obtido por meio dos resultados de garantia de qualidade em menor tempo (SILVA; ALVES; BRUNO, 2011).

Uma forma de diminuir os custos com atividade de automação de testes é consiste na realização de um levantamento dos requisitos e comportamentos esperados do sistema. Este levantamento permite priorizar a criação de casos de teste com maior probabilidade de encontrar defeitos no software, focando em pontos críticos e do sistema que necessitam de validação constante (FEWSTER; GRAHAM, 1999, tradução nossa; PRESSMAN; MAXIM, 2016).

Outro fator gera incertezas para as empresas que desejam realizar a implantação do processo de automação de testes é determinar se a atividade pode ser executada de forma bem-sucedida diante dos recursos disponíveis.

Por este motivo, é necessário que vários fatores sejam analisados para determinar a viabilidade da automação de testes. Estes fatores estão dispostos da seguinte maneira (FERRARI et al., 2009):

a) a maturidade e experiência da equipe responsável pela atividade de testes;
b) a necessidade de reutilização dos casos de teste, visto que a automação de testes não deve ser implementada para testes que ocorrem uma única vez;
c) a capacitação técnica da equipe de testes e o conhecimento das ferramentas de automação;
d) o entendimento dos requisitos e funcionalidades do sistema, incluindo os comportamentos e resultados esperados do sistema testado;

e) o tempo disponível para se investir na atividade de testes automatizados;

f) o conhecimento sobre a frequência de alterações das funcionalidades testadas, visando determinar o nível de manutenção que será necessário para os casos de teste;

g) a necessidade de alterações no código do sistema visando viabilizar a implementação de testes automatizados;

h) a validação dos casos de teste especificados com intuito de garantir que o mesmo nível de qualidade obtido por meio ao teste manual seja alcançado nos testes automatizados;

i) a disponibilidade da equipe para elaboração de um número considerável de casos de testes.

Entende-se que para uma execução efetiva da atividade de automação de testes é necessário determinar se o sistema possui um nível adequado de testabilidade, considerando os fatores que determinam a viabilidade da implantação e execução da automação de testes. (TANNOURI, 2015).

Além disso, para viabilização da automação de testes é necessário que haja a adoção de um modelo processo com intuito de organizar e gerenciar o processo de automação de testes. Este modelo deve seguir uma abordagem formada por etapas comuns em projetos de software como planejamento, analise, implementação e execução dos testes (RIOS; MOREIRA, 2006).

Durante as etapas de processo de testes, é importante que os haja a documentação dos requisitos funcionais do sistema, de forma detalhada, destacando os objetivos do sistema e de suas funcionalidades. Este documento servirá de base para realizar o desenvolver dos casos de teste manuais e automatizados (MIGUEL, 2015).

Além disso, é necessário que haja um estudo sobre os tipos de automação de testes para determinar a técnica que melhor se adequa ao projeto de software, e que poderá ser utilizado a para implementação e execução dos testes.

Segundo Molinari (2003) os tipos de automação de testes mais conhecidos podem ser definidos por:

a) ***capture-Playback:*** este método realiza a captura dos passos executados pelo usuário durante a interação com a interface do

sistema. São armazenados os dados de entrada e funcionalidades acessadas durante a atividade de testes;

b) ***data-Driven:*** esta abordagem realiza a execução dos testes por meio de *scripts* contendo dados isolados dos casos de teste. Os resultados destes testes são verificados utilizando os dados de uma base contendo as saídas esperadas. Esta abordagem possui um alto grau de reutilização e facilita a execução e manutenção dos testes;

c) ***keyword-Driven:*** foi desenvolvido para auxiliar na execução de testes de aceitação. A técnica assimila palavras chaves de linguagem natural a conjuntos de comandos de alto nível compreensíveis por pessoas sem conhecimento de programação de computadores. Estes comandos são executados por ferramentas de automação;

d) ***cli:*** abordagem técnica que permite a execução rotinas de teste por meio de um interpretador de linhas de comando. Esta técnica permite a interação com a aplicação por meio do sistema operacional.

Entende-se que a escolha do tipo de teste adequado para o projeto de software é imprescindível para uma execução com sucesso da atividade de automação de testes. Outro fator que é de extrema importância para uma implantação bem-sucedida da automação de testes é a escolha da ferramenta de automação que melhor se adapte as técnicas aplicadas e ao modelo de processo do projeto de software. No subcapítulo a seguir serão descritas as ferramentas utilizadas para a automação dos testes de software.

3.2 FERRAMENTAS DE AUTOMAÇÃO DE TESTE DE SOFTWARE

As ferramentas de automação possuem o objetivo de auxiliar na execução dos testes reduzindo o tempo necessário para empenhar esta atividade sem reduzir sua eficácia (PRESSMAN; MAXIM, 2016).

A utilização de ferramentas de software é indicada em ambientes onde existe a necessidade de validação do software em curtos prazos de tempo sem que haja impactos sobre a qualidade do produto. Também se recomenda a utilização de ferramentas para automação quando o perfil do sistema desenvolvido for de alta complexidade e sua funcionalidade impactar diretamente o negócio (BASTOS et al., 2007).

A seleção da ferramenta de automação correta é um importante passo para que se obtenha um processo efetivo de automação de testes. Diante disto, deve-se considerar as características das ferramentas de automação para que se possa determinar a ferramenta que melhor se adequa ao projeto de software desenvolvido. As principais características de ferramentas de automação podem ser consideradas da seguinte forma (SILVA; ALVES; BRUNO, 2011):

a) **complexidade**: o grau de complexidade envolvido para a utilização da ferramenta, visto que quanto maior a complexidade maior será o tempo necessário capacitação da equipe de testes;
b) **custo da ferramenta**: o custo da ferramenta utilizada para a automação de testes, visto que os retornos obtidos por meio do processo de automação de testes ocorrem à longo prazo, desta forma, é necessário cautela ao optar por ferramentas com alto custos em projetos que sem maturidade no processo de automação de testes e onde não há a garantia dos retornos;
c) **treinamento**: a especialização da equipe de testes e os treinamentos necessários para aprendizagem da aplicação devem ser levados em conta para a escolha da ferramenta de automação;
d) **reutilização de casos de teste**: a capacidade de reutilização dos casos de teste definidos;
e) **plataforma**: a plataforma a ser utilizada e as tecnologias suportadas.

Conclui-se que estas características permitem a equipe responsável pela escolha da ferramenta determinar a opção com maiores chances de aceitação, e compatibilidade com os recursos disponíveis para o projeto de software. É visto que a automação de testes possui variadas ferramentas que podem ser utilizadas para garantir a excelência do processo.

Estas ferramentas estão dispostas em vários formatos, complexidade, plataformas e licenças de uso. Nos subcapítulos a seguir estarão descritas as ferramentas *Calabash*, *Robotium* e *Appium* utilizadas para a automação de testes em aplicações móveis.

3.2.1 Calabash

Calabash é uma ferrramenta livre de custos e *open-source* desenvolvida para auxiliar na especificação e execução de casos de teste de aceitação para aplicações móveis. Esta ferramenta é mantida pela *Xamarin* e oferece suporte as plataformas Android e IOS (CALABASH, 2017, tradução nossa).

Segundo a *Xamarin (2017)*, os testes são escritos na linguagem *Cucumber* em associação com o conceito de *Behavior Driven Development (*BDD) para realizar a especificação dos casos de teste de aceitação, que de acordo com sua documentação contribui com o processo de automação das seguintes maneiras (CALABASH, 2017, tradução nossa):

a) facilita a comunicação entre desenvolvedores e especialistas por meio de uma linguagem única que minimiza os mal-entendidos à medida que a aplicação é desenvolvida;
b) a especificação dos testes fornece documentação clara e simples sobre como o sistema deve funcionar;
c) os testes possuem alta capacidade de validação regressiva, garantindo que alterações futuras não gerem impactos negativos no funcionamento do sistema;
d) a associação dos testes com o ambiente de integração contínua de desenvolvimento permite a validação constante dos requisitos, notificando os desenvolvedores a cada alteração de código se algo parar de funcionar no sistema.

Entende-se que a utilização da ferramenta *Calabash* possibilita a execução da atividade de automação de testes com foco no comportamento do sistema e nos resultados esperados a partir dos testes executados. Isto torna a atividade de especificação de casos de teste uma tarefa que pode ser executada por equipes que não possuam conhecimento a respeito de programação de computadores.

3.2.2 Robotium

O Robotium é uma ferramenta de automação de teste, que permite a especificação de casos de teste para aplicativos Android utilizando técnicas de caixa

preta e caixa branca. A ferramenta também permite a especificação de testes de aceitação, sistemas e usuários (KOCHHAR, 2015, tradução nossa*)*.

O framework de automação de testes Robotium possui características robustas que auxiliam o processo de testes. Estas funcionalidades podem ser citadas na seguinte ordem (NOGUEIRA, 2017):

a) testar aplicações *Android* nativas e híbridas;
b) requer um mínimo de conhecimento sobre os requisitos do sistema, possibilitando que a especificação dos casos de teste possam ser feitas por membros da equipe com menos experiência sobre as regras de negócio do sistema;
c) possibilita trabalhar com múltiplas instancias de aplicativos automaticamente;
d) viabiliza a especificação de scripts automatizados de forma rápida;
e) compreende-se melhor os casos de teste automatizado, uma vez que dispõe de comandos mais legíveis;
f) permite integrar o código com ferramentas de build como Maven, Gradle ou Ant para execução em integração contínua.

Além disso, o Robotium possui funcionalidades que permitem maior controle dos testes de software, por meio de recursos de (NOGUEIRA, 2017):

a) **captura de tela**: é necessário fornecer uma permissão de escrita no aplicativo, e para dispositivos virtuais, possuir um cartão de armazenamento montado. As capturas de tela são armazenadas no diretório */srcard/Robotium-Screenshots;*

b) **suporte a aplicações híbridas**: permite a interação com elementos de aplicações *web* a partir da versão 4.0 do Robotium;

c) **testes em aplicações pré-instaladas**: é possível realizar a execução de testes utilizando aplicativos instalados nativamente na plataforma *Android*. Para isso, é necessário que o aparelho móvel esteja desbloqueado.

Conclui-se que a ferramenta Robotium possui uma grande variedade de recursos que auxiliam a especificação e execução dos testes automatizados.

3.2.3 Appium

A ferramenta Appium possui licença *open-source* e gratuita para automação de aplicações móveis, *web* móveis e aplicações híbridas nas plataformas IOS e Android (APPIUM, 2017, tradução nossa).

O Appium foi projetado para atender às necessidades de automação móvel de acordo com uma filosofia delineada pelos seguintes princípios (APPIUM, 2017, tradução nossa):

a) durante o processo de automação, não deve haver a necessidade de recompilar seu aplicativo ou modificá-lo de forma alguma para automatizá-lo;
b) a especificação e execução dos testes não deve ser restrita a uma única linguagem de programação;
c) uma estrutura de automação móvel deve possibilitar a reutilização quando se trata de APIs de automação;
d) uma estrutura de automação móvel deve ter código aberto.

Entende-se que a ferramenta de automação Appium propõem um conceito de maior flexibilidade em relação a forma de execução e adaptabilidade para diversas plataformas e linguagens de programação.

A ferramenta possui uma arquitetura distinta que define a sua forma de funcionamento. Esta arquitetura permite a distribuição do trabalho de automação dos testes em diferentes módulos. Estes módulos podem ser definidos pelos seguintes conceitos (APPIUM, 2017, tradução nossa)*:*

a) **estrutura de cliente/servidor**: o envio e recebimento dos comandos é realizado por meio de transações de chamadas do tipo *Representational State Transfer* (REST)*.* Após estabelecer a conexão com o cliente, o servidor aguarda por comandos, executa estes comandos no aparelho móvel, e retorna uma resposta do tipo *Hypertext Transfer Protocol* (HTTP);
b) **sessão de comunicação**: no início da comunicação entre o cliente e o servidor é aberta uma sessão de comunicação, a qual é atribuído um identificador único. A sessão será responsável pela ligação entre o cliente e o servidor, garantindo que os comandos enviados sejam

entregues ao destino correto, e não para um outro aparelho na mesma rede de testes;

c) **configurações iniciais**: permite definir por meio de uma lista de chaves e valores enviados ao servidor Appium, as especificações desejadas para sessão de testes. É possível determinar qual plataforma será utilizada para execução dos testes, escolhendo entre Android e IOS, além de outras propriedades específicas como a ativação ou não do *Javascript.*

d) servidor: responsável pelo gerenciamento e execução dos comandos;

e) cliente: suporta várias linguagens de programação devido a forma de comunicação com o servidor ser baseada em HTTP, que é um protocolo altamente difundido.

Com base nas características abordadas pode-se entender que a estrutura de módulos da ferramenta Appium possui o objetivo de tornar acessíveis as suas funcionalidades para diversos tipos de sistemas e plataformas englobando protocolos que são utilizados pela maioria dos softwares.

Levando em consideração as características das ferramentas de automação abordadas neste subcapítulo, conclui-se que a ferramenta Appium se mostrou compatível com os objetivos propostos nesta pesquisa e será utilizada como meio de execução dos casos de teste automatizado aplicados durante este estudo.

4 TRABALHOS CORRELATOS

Para elaboração desta pesquisa foram abordados diferentes conceitos, técnicas e modelos de processos que contribuem com a atividade de testes e garantia de qualidade de software. Utilizou-se como parte da base de conhecimento para a realização desta pesquisa os trabalhos correlatos apresentados abaixo.

4.1 AUTOMAÇÃO DE TESTE PARA DISPOSITIVOS MÓVEIS E EXECUÇÃO DOS SCRIPTS DE TESTE AUTOMATIZADOS NA NUVEM

Esta monografia foi apresentada pelos acadêmicos Eduardo Corrêa de Sá e Rodrigo Silva do curso de Sistema da Informação, da Universidade do Sul de Santa Catarina – UNISUL, no ano de 2016, a fim de obter o título de bacharel em Sistemas da Informação, sob orientação do Prof. Dr. Saulo Popov Zambiasi.

O objetivo do trabalho é realizar um levantamento de ferramentas de automação utilizadas para a garantia da qualidade de software em dispositivos móveis. As ferramentas comparadas durante a pesquisa foram: Selenium (*Framework* de automação de testes), Appium (*Framework* de automação de testes *mobile*), TestNG (*Framework* de execução de testes) e SauceLabs (Serviço de emulação de dispositivos móveis na nuvem). O estudo também visa comparar as técnicas e os conceitos utilizados durante o processo de automação de testes.

A pesquisa aborda as relações métricas quantitativas obtidas por meio dos relatórios de testes e qualitativas por meio da análise e comparação dos dados dos dados gerados pelo processo de automação. Esta pesquisa demonstrou que a execução de testes automatizados por meio da utilização de emuladores para dispositivos moveis é mais lenta do que quando comparado a execução em dispositivos reais, contudo, esta ferramenta se mostrou eficaz no processo de garantia de qualidade de software, visto que a mesma pode servir de auxílio para a execução de testes automatizados em situações que não há a existência de dispositivos móveis reais.

4.2 GERAÇÃO AUTOMÁTICA DE CASOS DE TESTE AUTOMATIZADO NO CONTEXTO DE UMA SUITE DE TESTES EM TELEFONES CELULARES

Pesquisa elaborada pelo acadêmico Bruno Martins Petroski do curso de Ciência da Computação, da Universidade de Federal de Santa Catarina – UFSC, no ano de 2006, a fim de obter o título de Bacharel em Ciências da Computação, sob orientação do Prof. D.Sc. Douglas Nascimento Rechia e coorientação do Prof. D.Sc Ricardo Pereira e Silva.

A pesquisa evidencia a importância dos casos de teste automatizado e do teste exploratório para a garantia da qualidade de software. Contudo, a pesquisa demonstra que existe uma dificuldade na criação de casos de teste que atendam a esta abordagem exploratória dos testes. Buscando amenizar este problema, o trabalho proposto pelo acadêmico apresenta um método que visa possibilitar a geração automática de casos de teste automatizado por meio de *logs* registrados durante sessões de testes exploratórios. O intuito da pesquisa é viabilizar a incorporação destes casos de teste ao ciclo de vida do processo de execução de testes, permitindo que estes testes garantam que erros encontrados durante os testes exploratórios não voltem a ocorrer após a sua correção.

4.3 GERAÇÃO DE CASOS DE TESTE FUNCIONAL PARA APLICAÇÕES DE CELULARES

Monografia apresentada pela acadêmica Emanuela Gadelha Cartaxo do curso de Pós-Graduação em Informática da Universidade Federal de Campina Grande – UFCG, no ano de 2006, a fim de obter o título de Mestre em Ciência da Computação, sob orientação da Prof. Patrícia Duarte de Lima Machado.

O trabalho elaborado propõe um procedimento que visa permitir a geração de casos de teste funcionais com foco em *features* do sistema.

A pesquisa é direcionada para a aplicação de testes em dispositivos móveis que utilizem a abordagem sistemática de testes baseados em diagramas de UML. O trabalho evidencia a importância da derivação de testes a partir destes diagramas e que apesar de contribuir com o processo de testes não é plenamente abordada.

4.4 O USO DAS TÉCNICAS DATA-DRIVEN E KEYWORD-DRIVEN NO PROCESSO DE DESENVOLVIMENTO DOS SCRIPTS DE AUTOMAÇÃO DE TESTES FUNCIONAIS DE SOFTWARE

Trabalho de conclusão de curso realizado pelo acadêmico Marcelo Dehon Batista de Prá Souza do curso de Ciência da Computação, da Universidade do Extremo Sul Catarinense – UNESC, no ano de 2012, a fim de obter o título de Bacharel em Ciência da Computação, sob orientação do Prof.ª. MSc. Ana Claudia Garcia Barbosa.

O trabalho proposto visa demonstrar as técnicas de automação de testes e técnicas de *data-driven* utilizadas para criação de scripts de execução de testes. A pesquisa reúne as principais técnicas de automação implementadas pelo mercado fornecendo uma visão sobre a geração de casos de teste. O objetivo da pesquisa é servir como base para criação de teste de software que realizados de forma automatizada, com foco na abordagem de testes regressivos. O trabalho visa o entendimento da metodologia aplicada em processos de desenvolvimento, tratando das estratégias utilizadas para validação de software e garantia de qualidade.

5 PROPOSTA PARA MELHORIA NA CRIAÇÃO DE TESTES AUTOMATIZADOS UTILIZANDO OS REGISTROS DAS ATIVIDADES DOS USUÁRIOS DE APLICAÇÕES MÓVEIS

Esta pesquisa é um procedimento que consiste no desenvolvimento de uma biblioteca de criação de testes automatizados para aplicações móveis. Propõe-se por meio desta pesquisa uma melhoria na criação de casos de teste automatizado minimizando o problema da falta de cobertura de testes exploratórios para aplicações móveis.

Pretende-se alcançar essa melhoria por meio da disponibilização de recursos e métodos de registro de atividades desenvolvidos nesta pesquisa. Utilizando-se da biblioteca de criação de testes busca-se possibilitar a geração de novos casos de teste automatizado utilizando os registros das atividades. Além disso, busca-se possibilitar que as equipes responsáveis pela qualidade do teste de software tenham acesso a este recurso auxiliando na etapa de especificação de casos de teste automatizado.

No desenvolvimento deste trabalho foram empregados algoritmos de teste automatizado para aplicações móveis. Foram gerados casos de teste automatizado a partir de algoritmos de teste automatizado. Realizou-se a implementação de uma aplicação móvel de testes com intuito de empregar os algoritmos de teste automatizado.

Os resultados deste estudo foram obtidos com base na análise feita a partir dos casos de teste gerados considerando a sua capacidade de reproduzir as atividades dos usuários e realizar testes exploratórios.

5.1 METODOLOGIA

Para que os objetivos do trabalho fossem atingidos, realizou-se as seguintes etapas metodológicas: levantamento bibliográfico, investigação sobre algoritmos de teste automatizados, implementação de um algoritmo de registro de atividades, implementação de um algoritmo de geração de casos de teste automatizado, aplicação dos algoritmos em uma aplicação móvel de testes, análise dos casos de testes gerados a partir dos algoritmos implementados e geração de hipóteses sobre testes automatizados.

A pesquisa bibliográfica proporcionou que na fundamentação teórica fossem abordados conceitos sobre a garantia de qualidade de software, incluindo as estratégias e etapas que constituem o processo de teste de software. Por último, é feito uma abordagem sobre ferramentas de automação para aplicações móveis, pelo qual nesta pesquisa optou-se pela utilização da ferramenta Appium.

5.1.1 Investigação dos algoritmos de teste automatizado para aplicações móveis

Durante a elaboração desta pesquisa foram realizados estudos sobre algoritmos de teste automatizado para aplicações móveis. Foram especificados alguns testes automatizados com intuito de compreender mais sobre esses algoritmos e coletar dados sobre o tempo investido durante a especificação manual de casos de teste automatizado. Este estudo proporcionou conhecimento sobre o funcionamento destes algoritmos e sua forma de especificação.

Compreendeu-se que algoritmos de teste automatizado possibilitam a reprodução de determinadas atividades de interação com aplicações móveis. As atividades são reproduzidas por meio execução de comandos de programação. Estes comandos representam atividades de uma aplicação móvel incluindo cliques e o pressionamento de teclas.

Para o desenvolvimento da parte prática desta pesquisa realizou-se um estudo sobre os *frameworks* de teste automatizado Appium, Calabash e Robotium. Estes *frameworks* estão relacionados com o conteúdo abordado na subseção 3.2, onde são estudadas as ferramentas de automação utilizadas por estes *frameworks* de teste.

Para alcançar o objetivo geral proposto por esta pesquisa, optou-se por realizar a especificação dos algoritmos de teste automatizado utilizando o *framework* de testes Appium. Levou-se em consideração os recursos disponibilizados por este *framework* de testes e a sua compatibilidade com a ferramenta de automação Appium, visto que a mesma foi criada para execução dos algoritmos do Appium.

5.1.1.1 Recursos da API de testes do *framework* Appium

Os recursos disponibilizados pelo *framework* de teste Appium se mostraram fundamentais para suprir as necessidades do desenvolvimento da parte prática desta pesquisa. Podem ser citados como recursos utilizados: a localização de elementos da interface do sistema por meio de anotações, a integração com o *framework* UIAutomator e a disponibilidade de algoritmos que simulam interações com a interface do sistema.

A localização de elementos da interface do sistema por meio da anotação *AndroidFindBy* se mostrou um recurso importante para o desenvolvimento deste estudo, pois possibilitou aos algoritmos de teste automatizado delegar ao *framework* de testes a tarefa de verificação da existência de elementos de interface durante a navegação entre as telas do sistema.

Figura 3 – Especificação da anotação *AndroidFindBy*

```
//Anotação para localizar o elemento de interface
@AndroidFindBy(id = "nomeMotorista")
AndroidElement nomeMotorista;

@Test
public void teste() {

    //Acesso ao elemento sem necessidade de verificação de existência
    nomeMotorista.click();

}
```

Fonte: Do autor.

Na figura 3 é possível visualizar a especificação do caso de teste utilizando a anotação *AndroidFindBy* para localizar o elemento de interface.

No acesso aos elementos de interface esse recurso realiza a verificação de existência do elemento, evitando a acesso a uma variável com valor nulo.

Este recurso também permite que seja definido um tempo limite para localização dos elementos de interface, evitando que haja o acesso precoce destes elementos antes da interface finalizar seu carregamento, causando falhas em testes, que podem vir a ser incorretamente interpretadas como defeitos no sistema.

O recurso de integração com o *framework* nativo de testes da plataforma Android UIAutomator possibilitou a esta pesquisa a utilização de localizadores de elementos de interface com filtros de busca. Estes localizadores foram acessados meio da classe *AndroidElement* realizando a chamada do método *findElementByAndroidUIAutomator.* Este método recebe como parâmetro um objeto *UiSelector*, que possibilita realizar a busca do elemento na hierarquia de elementos levando em consideração as condições especificadas como critérios de busca (ANDROID, 2018).

O recurso de algoritmos de simulação de interações com interfaces auxiliou este estudo na reprodução de ações mais complexas para serem executadas por algoritmos convencionais de automação de testes. As ações incluem a simulação do movimento de arrastar os dedos enquanto pressiona um elemento de interface e toques em pontos predeterminados da tela, conhecidos como *tap*. O estudo destes recursos foi fundamental para ampliar o conhecimento sobre algoritmos de automação e as estratégias disponíveis para aprimorar o processo de especificação de algoritmos de automação.

5.1.1.2 Estrutura dos algoritmos de teste automatizado

O estudo da especificação dos casos de teste automatizado proporcionou o conhecimento da estrutura dos algoritmos de teste automatizado. Esta estrutura possibilitou organização do algoritmo de forma adequada delimitando o ciclo de vida do teste automatizado.

Para a base desta estrutura, cria-se uma classe de algoritmos de automação. Nesta classe especifica-se o método inicial de configuração dos testes, chamado de *setup*. Este método foi marcado por meio da anotação *Before*.

No método *setup*, define-se as variáveis de preparação dos testes automatizados. Essas variáveis indicam as configurações mínimas para a execução dos testes.

Tabela 1 – Variáveis do Método *Setup*

Função	Tipo	Descrição
driver	AndroidDriver	*Driver* de execução dos testes
appiumServerAddress	String	Endereço do servidor Appium
PLATFORM_VERSION	String	Versão do sistema móvel operacional
DEVICE_NAME	String	Nome do dispositivo
PLATFORM_NAME	String	Nome do sistema operacional
APP	String	Pacote de instalação da aplicação

Fonte: Do autor.

Na tabela 1 é possível visualizar a configuração utilizada durante a elaboração desta pesquisa. Esta configuração foi implementada de acordo com o ambiente de testes preparado.

O método utilizado para especificação do teste recebeu a anotação *Test*. Neste método foram especificados os passos que formam o caso de teste dentro do algoritmo de teste automatizado. Estes passos consistem em uma sequência de comandos e asserções provenientes do *framework* de teste Appium.

Por último, o método de encerramento dos testes foi denominado de *tearDown*, e foi marcado utlizando a anotação *After.* Neste método definiu-se as ações de encerramento do teste que independem do resultado das asserções do caso de teste.

Figura 4 – Estrutura do algoritmo de teste automatizado

```
public class TestePrincipal {

    @Before
    public void setup() {

    }

    @Test
    public void teste() {

    }

    @After
    public void tearDown() {

    }
}
```

Fonte: Do autor.

Na figura 4 é possível visualizar a especificação mínima da estrutura de um algoritmo de teste automatizado e todos os elementos que compõem a sua estrutura. Esta estrutura representa o ciclo de vida do teste automatizado, desde o a preparação até a finalização do teste.

O sequenciamento do teste por meio desta estrutura permite a implementação de comportamentos genéricos durante a execução dos testes. Estes comportamentos consistem na inicialização, execução e termino do teste.

Figura 5 – Especificação do comando *sendKeys*

Fonte: Do autor.

A figura 5 ilustra a especificação de um de teste automatizado contendo o comando *sendKeys* utilizado para pressionar teclas em um campo. A forma de especificação dos algoritmos está diretamente ligada ao *framework* de testes utilizado.

O estudo sobre os algoritmos de teste automatizado abordado neste capítulo forneceu o embasamento teórico necessário para o desenvolvimento dos algoritmos de registro de atividades e geração de casos de teste automatizado. Estes algoritmos foram desenvolvidos como parte da solução proposta para o problema apresentado nesta pesquisa. No decorrer deste capítulo serão abordados com maiores detalhes os algoritmos de registro de atividades e geração de casos de teste automatizado.

5.1.2 Algoritmo de registro de atividades

Nesta etapa iniciou-se o desenvolvimento do algoritmo de registro de atividades de usuários de aplicações móveis.

O objetivo deste algoritmo é registrar as atividades que dificilmente são consideradas durante a etapa de especificação de casos de teste automatizado. Optou-se por utilizar estas atividades como base para melhoria da criação de casos de teste exploratório.

Para delimitar um escopo de estudo optou-se por realizar o registro das seguintes atividades executadas em aplicações móveis: navegação entre as telas, entrada de valores em campos, foco em campos, pressionamento de teclas, seleção de opções em caixas de opções, seleção de itens em listagens, alteração de estados de caixas de seleção, movimentação de barras de progresso e cliques em botões.

A escolha destas atividades se mostrou satisfatória para alcançar o objetivo proposto pelo algoritmo de registro de atividades, pois envolvem diferentes tipos de rotinas aumentando a diversidade da forma de execução para cada uma destas atividades.

5.1.2.1 Especificação do algoritmo de registro de atividades

O algoritmo de registro de atividades tem sua especificação contida em uma classe denominada *RegistroAtividades*. Na tabela 2 têm-se os métodos implementados relativos ao registro das atividades:

Tabela 2 – Métodos da Classe *RegistroAtividades*

Função	Tipo	Descrição
inicializar	void	Inicializa as variáveis
vincularElemento	Atividade	Vincula o elemento de interface a uma instancia da classe *Atividade*
registrarAcessoTela	void	Registra o nome da tela acessada pelo usuário
pressionar	void	Simula o pressionamento de uma tecla física do aparelho móvel
adicionarAtividade	void	Adiciona uma atividade na lista de atividades registradas
getListaAtividades	ArrayList<Atividade>	Retorna a lista de atividades registradas
getINSTANCE	RegistroAtividades	Retorna uma instancia da classe *RegistroAtividades*

Fonte: do autor

Considerando o método *vincularElemento* apresentado na tabela 2 que realiza a criação do vínculo entre o elemento de interface e uma instância da classe *Atividade*.

Figura 6 – Especificação do método *vincularElemento*

```
public static Atividade vincularElemento(View elemento) {
    String identificador = ElementIdParserUtilitario.getStringId(elemento);
    Atividade atividade = new Atividade(INSTANCE);
    MetodosUtilitario.invocarMetodo(atividade,  methodName: "setIdentificadorElemento",
            String.class, identificador);
    return atividade;
}
```

Fonte: Do autor.

Na figura 6 ilustra-se o a especificação do método *vincularElemento* que retorna um objeto do *Atividade* a qual o elemento de interface foi vinculado por meio do método *setIdentificadorElemento*.

A classe *Atividade* disponibiliza métodos de registro de atividade que permitem identificar a atividade executada pelo usuário e realizar o seu registro de forma devida. Estes métodos estão dispostos na tabela 3.

Tabela 3 – Métodos de registro de atividade

Função	Tipo	Descrição
selecionarOpcao	Atividade	Registra atividade de seleção de opção em caixas de opções
selecionarItemListagem	Atividade	Registra atividade de seleção de item em listagens
focarCampo	Atividade	Registra atividade de foco em elementos de interface
limparValor	Atividade	Registra atividade de remoção de valores de elementos de interface
escreverValor	Atividade	Registra atividade de entrada de valores de elementos de interface
lerValor	Atividade	Registra atividade de asserção de valor contido em um elemento de interface
rolarAteCampo	Atividade	Registra atividade de navegação até determinado elemento de interface
clicarBotao	Atividade	Registra atividade de clique em um elemento de interface
desfocarCampo	Atividade	Registra atividade de remoção do foco de elemento de interface
deslizarBarraProgresso	Atividade	Registra atividade de alteração do progresso de barras de progresso
marcarOpcaoDesmarcavel	Atividade	Registra atividade de marcação de opções desmarcáveis
marcarOpcao	Atividade	Registra atividade de marcação de opções não desmarcáveis
pressionarTeclas	Atividade	Registra atividade de pressionamento de teclas em um elemento de interface
reproduzirAcoes	Encerramento	Adiciona a atividade de reprodução a listagem de atividades
verificarValores	Encerramento	Adiciona a atividade de asserção a listagem de atividades

Fonte: do autor

Os métodos apresentados na tabela 3 realizam o armazenamento do tipo da atividade e das informações necessárias para sua reprodução. Quando atividade é encerrada o método *reproduzirAcoes* adiciona esta atividade a lista de atividades registradas.

A lista das atividades registradas durante a execução do algoritmo de registro de atividades pode ser obtida por meio do método *getListaAtividades*.

A listagem é utilizada pelo algoritmo de geração de casos de teste automatizado para a criação de novos casos de teste automatizado.

A utilização do algoritmo de registro de atividades busca contribuir com processo de teste de software por meio do aumento da capacidade exploratória dos casos de teste automatizado e da diversidade na forma de especificação dos casos de teste.

5.1.3 Algoritmo de geração de casos de teste automatizado

Nesta etapa, iniciou-se o desenvolvimento do algoritmo de geração de casos de teste automatizado.

Propõem-se por meio deste algoritmo uma melhoria na criação de casos de teste automatizado, utilizando-se das atividades registradas pelo algoritmo de registro de atividades. Esta abordagem busca ampliar a cobertura de casos de teste exploratórios aumentando a semelhança com o contexto de uso dos usuários.

5.1.3.1 Especificação do algoritmo de geração de casos de teste automatizado

O algoritmo de geração de casos de teste automatizado foi especificado por meio da classe *GeradorCasosTeste*. Neste algoritmo estão dispostos métodos que permitem a criação de um caso de teste a partir de registros de atividades. A tabela 4 demonstra estes métodos:

Tabela 4 – Métodos da classe *GeradorCasosTeste*

Função	Tipo	Descrição
inicializar	void	Inicializa as variáveis
gerar	Teste	Gera o caso de teste

Fonte: Do autor.

Por meio da tabela 4 pode-se observar o método *inicializar* que recebe como parâmetro uma variável do tipo *Setup* onde são especificadas as configurações para execução do caso de teste automatizado. O segundo parâmetro deste método permite o envio de uma variável do tipo *Preferencias* que define as características de especificação do caso de teste automatizado. Por meio da

especificação deste parâmetro é possível definir as seguintes características: pacotes de importação, nome do pacote de testes, nome da classe pai, declaração de métodos específicos, declaração do método *tearDown*. As variáveis *setup* e *preferencias* são fundamentais para o funcionamento do método *gerar.*

O método *gerar* recebe como parâmetro uma lista de objetos do tipo *Atividade*. A lista de atividades é convertida em um algoritmo de teste automatizado. O segundo parâmetro do método *gerar* é dado pelo nome dado ao caso de teste.

Figura 7 – Especificação do método *gerar*

```
public static Teste gerar(ArrayList<Atividade> atividades, String nomeTeste) {

    GeradorCasosTeste.nomeTeste = nomeTeste == null ? GeradorCasosTeste.nomeTeste : nomeTeste;
    GeradorCasosTeste.atividades = atividades;
    Teste teste = montadorTestes.montarTesteCompleto();
    fullScript = montadorTestes.montarSetup();
    fullScript += teste.getCasoTeste();
    if (!GeradorCasosTeste.preferencias.isSkipTearDownDeclaration()) {
        String teardownScript;
        teardownScript = montadorTestes.montarTeardown();
        fullScript += teardownScript;
    }
    fullScript = montadorTestes.montarClasseTeste();
    fullScript = fullScript.replace( target: "\n\n\n\n",  replacement: "\n\n");


    teste.setCasoTeste(fullScript);
    inicializar(GeradorCasosTeste.setup, GeradorCasosTeste.preferencias);

    return teste;
}
```

Fonte: Do autor.

Na figura 7 é possível observar a especificação do método *gerar*. Em sua definição o método *gerar* realiza a execução de funções internas de classes aninhadas. As classes *MontadorTestes*, *MontadorMetodos* realizam a construção do caso de teste automatizado utilizando as informações contidas nas variáveis *setup*, *preferencias, nomeTeste* e *atividades*. A classe *MontadorDocumentacao* utiliza as atividades registradas para montar a documentação do caso de teste.

O resultado da execução do método *gerar* é um objeto do tipo *Teste* que contém o caso de teste automatizado e a documentação dos passos reproduzidos pelo teste. O teste gerado por este algoritmo possui a mesma estrutura abordada na subseção 5.1.1.2.

A utilização do algoritmo de geração de casos de teste automatizado busca possibilitar ao processo de teste de software alcançar um nível maior de

conformidade entre os casos de teste e a documentação especificada. Isto contribui com o processo de garantia de qualidade de software, e aumenta a credibilidade e confiabilidade do sistema.

5.1.4 Aplicação do conteúdo desenvolvido

Os algoritmos de registro de atividades e geração de casos de teste automatizado foram aplicados em uma aplicação móvel de testes durante o desenvolvimento deste estudo. A aplicação móvel foi desenvolvida para que se pudesse realizar a simulação de um usuário durante a atividade de interação enquanto realiza-se o registro das atividades e geração de casos de teste automatizado.

5.1.4.1 Desenvolvimento da Aplicação móvel de testes

A aplicação móvel de testes consiste em um sistema que permite o cadastro e consulta de registros de motoristas. Criou-se uma lista de requisitos de interface as quais deseja-se atender por meio do desenvolvimento da aplicação móvel de testes. Busca-se por meio destes requisitos possibilitar a aplicação dos algoritmos de registro de atividade e geração de casos de teste automatizado de forma devida.

Pode-se citar os seguintes requisitos que a aplicação móvel de testes atende: navegação das telas, elementos de interface com conteúdo editável, botões de ação, listagem de itens, barras de progresso, caixas de seleção e caixas de opções. Realizou-se a implementação de duas telas possibilitando ao usuário navegar na aplicação ampliando as possibilidades de variações das atividades executadas.

No primeiro acesso a aplicação o usuário tem a visão da tela de consulta de registros. Neste momento a instrução que indica o início do registro das atividades é disparada.

A partir deste momento, todas as atividades realizadas utilizando a aplicação móvel são consideradas para realizar a criação do caso de teste automatizado.

Utilizou-se dos eventos de interface disponibilizados pelo Android para vincular o algoritmo de registro de atividades. Esses eventos permitem que seja acionada uma função no momento em que há interação com um elemento de interface.

Figura 8 – Tela de Consulta

Fonte: Autor (2018)

Na figura 8 é possível visualizar a primeira tela da aplicação móvel de testes, a tela de consulta de registros.

Nesta tela é possível realizar buscas entre os registros existentes filtrando pelo nome do registro. Além disso, é possível utilizar essa tela para acessar registros inseridos na memória da aplicação móvel de testes.

O encerramento do registro das atividades também foi inserido nesta tela, no momento em que o usuário clica no botão de copiar, a aplicação encerra o registro de atividades e realiza a chamada do método que gera um novo caso de testes.

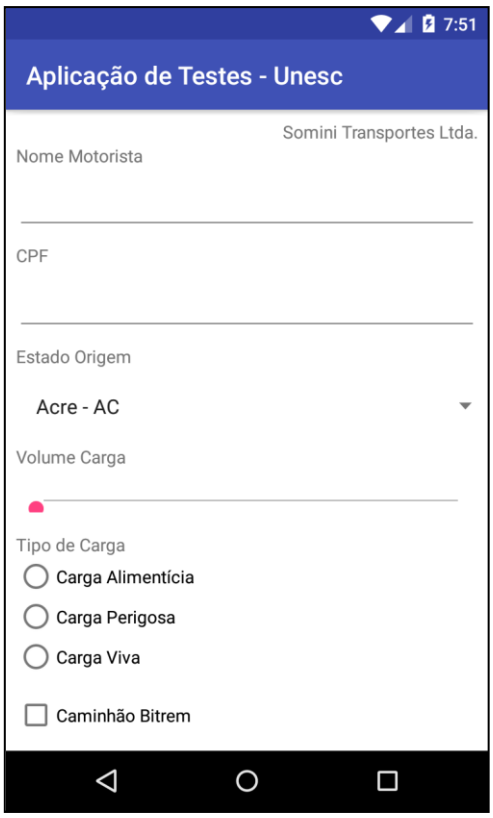
Procurou-se realizar implementação de uma segunda tela para possibilitar a navegação na aplicação.

O acesso a essa segunda tela é feito por meio do botão de adicionar que esse encontra na tela de consulta. Quando o usuário clica para acessar a tela de

cadastros a aplicação registra um acesso de uma nova tela e continua o registro das atividades.

No momento em que o usuário acessa a tela de cadastro, ele encontra uma tela com elementos de interface que possibilitam a inserção de um novo registro.

Figura 9 – Tela de Cadastro



Fonte: Autor (2018)

Na figura 9 é possível observar a segunda tela de cadastro de motoristas. Ao acessar a tela de cadastro pode-se observar os elementos de interface utilizados para realizar o cadastro de um novo registro. Buscou-se realizar a inclusão de elementos de interface de tipos variados com intuito de validar os diferentes aspectos da interação dos usuários de aplicações móveis.

5.1.4.2 Aplicação dos algoritmos dos algoritmos implementados para criação de casos de teste automatizado

Os algoritmos de registro de atividades e geração de casos de teste automatizado foram armazenados em uma biblioteca do tipo *AAR*. Esta biblioteca foi

publicada no repositório público *JCenter* possibilizando a sua utilização como uma dependência de projetos de aplicações móveis.

A publicação da biblioteca em um repositório online possibilita o acesso por meio do projeto da aplicação móvel de testes. A importação da biblioteca de testes foi realizada a partir da inserção de uma linha de código no arquivo *build.gradle* do projeto da aplicação móvel de testes.

Figura 10 – Importação da biblioteca de testes

```
dependencies {

    implementation 'appiumrunner:unesctcc3:6.09.22.18'

}
```

Fonte: Do autor.

A figura 10 é ilustra a importação da biblioteca de testes referenciando uma versão específica. Ao importar a dependência no projeto inicia-se um processo construção executado pelo ambiente de desenvolvimento integrado Android com intuito de obter os arquivos que contém os algoritmos de registro de atividade e geração de casos de teste automatizado.

Após a importação das dependências no projeto, é possível realizar a aplicação do algoritmo de registro de atividades. A aplicação do código do algoritmo de registro de atividades é feita por meio da inserção de uma instrução no código da aplicação móvel de testes. Para isto, mapeiam-se os eventos nativos da aplicação móvel de testes onde deseja-se realizar a inserção da instrução.

Neste estudo foram inseridas instruções em eventos nativos da aplicação móvel de testes buscando abranger todas as áreas do sistema. A especificação das instruções de registro de atividades é fundamental para que as atividades sejam registradas no momento devido.

Especifica-se em um primeiro momento a instrução que indica o início do processo de registro das atividades. Esta instrução consiste na especificação do método *inicializar*.

Em seguida, realiza-se a chamada do método *vincularElemento* indicando o elemento de interface ao deve-se realizar o vínculo do algoritmo de registro de atividades.

Figura 11 – Especificação do comando *vincularElemento*

```
RegistroAtividades.vincularElemento(nomeMotorista);
```

Fonte: Do autor.

A figura 11 ilustra a especificação do comando *vincularElemento* utilizando o elemento de interface *nomeMotorista.* A execução do método *vincularElemento* retorna um objeto do tipo da classe *Atividade.* A partir deste retorno realiza-se a especificação de um método de registro de atividade. Os métodos de registro de atividades disponíveis estão documentados na tabela 3 da subseção 5.1.2.1.

A especificação destes métodos é realizada de forma encadeada, onde o resultado de uma chamada retorna o mesmo tipo de objeto *Atividade*. Isto possibilita que para o mesmo elemento a chamada de registro de atividades seja feita apenas uma vez.

Figura 12 – Chamada de um método de registro de atividades

```
RegistroAtividades.vincularElemento(nomeMotorista)
        .focarCampo()
        .escreverValor(text)
        .reproduzirAcoes()
        .verificarValores();
```

Fonte: Do autor.

Na figura 12 pode-se observar a chamada dos métodos de registro de atividade *focarCampo* encadeando o método *escreverValor.* Utiliza-se esses métodos para registar as atividades de foco e entrada de valores no elemento de interface *nomeMotorista* localizado na tela de cadastro. No final do método de registro de atividade realiza-se o a especificação do método *reproduzirAcoes* que adiciona as atividades registradas à lista de atividades. E o método *verificarValores* adiciona uma atividade assertiva a lista de atividades.

O encerramento do registro das atividades é realizado por meio da especificação do método *gerar* da classe *GeradorCasosTeste*. Este método recebe as atividades registradas realizando a criação de um caso de teste automatizado e gerando a documentação para os testes criados.

Para obter o caso de teste e a documentação dos testes gerados deve-se armazenar o retorno do método *gerar* em uma variável do tipo *Teste*. É de responsabilidade da equipe de testes gerar o arquivo de teste com extensão *.java* onde será inserido o caso de teste.

5.1.5 Aplicação prática da pesquisa em ambientes de teste

A aplicação desta pesquisa em um ambiente de teste proporcionou uma perspectiva dos possíveis desafios e as vantagens consequentes da utilização deste procedimento.

Optou-se realizar a aplicação desta pesquisa por meio da execução das etapas que compõem o ciclo de vida do teste de software abordadas na subseção 2.1.2 com objetivo de demonstrar a integração entre os processos convencionais de automação de testes e a aplicação deste estudo.

5.1.2.1 Procedimentos iniciais

Na etapa de procedimentos iniciais realizou-se uma análise da aplicação móvel de testes com intuito determinar a sua maturidade mediante a aplicação de uma abordagem de testes de sistema, tratado na subseção 2.2.3. Buscou-se determinar a integridade do sistema por meio de execução de rotinas que se assemelham a forma de utilização do usuário final.

Determinou-se a utilização da técnica de teste caixa preta com foco no teste de regressão, abordado na subseção 2.3.2, com o objetivo de garantir que as funcionalidades já testadas prossigam funcionando após alterações realizadas na aplicação. Isto vale da mesma forma para o teste automatizado, que é reexecutado inúmeras vezes durante o processo de garantia de qualidade de software garantindo assim a conformidade do sistema.

Buscou-se determinar a viabilidade da implantação do processo de automação de testes por meio dos fatores considerados na subseção 3.1, onde

aborda-se as precauções que devem ser tomadas ao considerar a implantação de testes automatizados.

Definiu-se quais funcionalidades da aplicação móvel de testes deseja-se pela aplicação dos algoritmos de registro de atividades e geração de casos de teste automatizado. Por fim, deu-se início ao planejamento dos casos de teste.

5.1.2.2 Planejamento

O planejamento da aplicação dos algoritmos de registro de atividades e geração de casos de teste automatizado foi feito com base em uma estratégia de testes exploratórios, tratados na subseção 2.3.2.1. Buscou-se desta forma a abranger o maior número de áreas possíveis dentro da aplicação móvel de testes.

Para determinar os pontos de aplicação dos algoritmos, considerou-se a maturidade de cada funcionalidade, e a probabilidade de revelar defeitos no sistema.

Determinou-se o autor desta pesquisa como responsável pelo papel de usuário da aplicação móvel de testes. Com base nessa informação, foi realizada uma estimativa do tempo que seria investido na execução da aplicação, buscando estabelecer um tempo limite para fins de planejamento e cumprimento do cronograma disponível.

Foram atribuídas as responsabilidades ao usuário de utilizar a aplicação durante o processo de registro das atividades. Busca-se com esta atividade realizar a execução da aplicação móvel de testes com base na experiência adquirida por meio de experiências e resultados de baterias de teste anteriores.

5.1.2.3 Preparação

A etapa de preparação foi realizada em paralelo com as demais etapas do ciclo de vida do teste de software com objetivo de agilizar o processo de preparação. Nesta etapa realizou-se a verificação dos recursos necessários a execução dos testes. Realizou-se a configuração do emulador de dispositivos móveis para plataforma Android, efetuou-se a instalação e configuração da aplicação móvel de testes no emulador, fez-se a configuração da ferramenta de automação Appium e da IDE Android Studio.

Certificando-se que todas os recursos estivessem preparados, realizou-se um teste mínimo de funcionamento, para garantir que ao iniciar a execução da aplicação móvel de testes não houvessem interrupções por não conformidades na configuração dos recursos.

5.1.2.4 Especificação

Na etapa de especificação realizou-se a aplicação dos algoritmos de registro de atividades e geração de casos de teste automatizado na aplicação móvel de testes. Neste momento, realizou-se a aplicação dos métodos de registro de atividades nos eventos de elementos de interface.

Em um modelo convencional de ciclo de vida de teste de software realiza-se a especificação e documentação dos passos a serem seguidos em cada caso de teste, conforme visto na subseção 2.1.2.

Contudo, como visto anteriormente na subseção 5.1.4.2, a aplicação dos algoritmos desenvolvidos durante esta pesquisa diminui a necessidade de execução desta etapa, pois possibilita a geração automática dos casos de teste e da documentação dos testes.

Ainda assim, deve-se ter em mente que a etapa de especificação não pode ser totalmente excluída do ciclo de vida de teste de software. Mantém-se a necessidade de determinada etapa para que as equipes responsáveis pela manutenção dos testes possam validar a conformidade dos casos gerados por meio desta pesquisa.

Além disso, é valido ressaltar, que as variações dos casos de teste tendem a se repetir a medida que a base de usuários da aplicação cresce. Por isso, é importante que haja uma equipe responsável pela validação destas não conformidades.

5.1.2.5 Execução

Na etapa de execução da aplicação móvel de testes buscou-se realizar a execução das atividades de forma coerente, seguindo fluxos lógicos. Optou-se por rotinas com características de execução de testes de sistema, e garantia das conformidades realizadas em testes de usabilidade. Em um segundo momento,

foram realizadas variações de aspecto exploratório. Executou-se diversas vezes as mesmas atividades intercalando entre comportamentos e telas do sistema e formas de execução.

Durante toda a etapa de execução foram registradas as atividades realizadas pelo usuário. As atividades registradas caracterizam o caso de teste e a documentação gerados a partir dos algoritmos de registro de atividades e geração de casos de teste automatizado.

5.1.2.6 Entrega

Nesta etapa finalizou-se a bateria de testes com a entrega do caso de teste automatizado e a documentação dos testes. Esta etapa foi documentada por meio do armazenamento do teste gerado em um arquivo de extensão *.java* e a documentação do teste foi armazenada em um arquivo de texto. Espera-se que o conteúdo obtido por meio da execução dos testes indique a reprodução das rotinas realizadas pelo usuário.

Tendo concluído a execução das etapas do ciclo de vida do teste de software, iniciou-se uma análise do caso de teste gerado a fim de determinar a sua cobertura exploratória.

Realizou-se a execução do caso de teste para garantir que as atividades executadas pelo usuário foram replicadas para o caso de teste automatizado. Utilizou-se dos resultados dos testes para determinar a conformidade entre as atividades executadas, o caso de teste automatizado. A documentação gerada a partir do algoritmo de geração de casos de teste automatizado também foi considerada durante esta análise.

Os resultados obtidos durante a aplicação e execução desta pesquisa serão discutidos no decorrer deste capítulo com intuito de determinar o êxito na obtenção dos resultados esperados e dos objetivos propostos por meio deste estudo.

5.2 RESULTADOS OBTIDOS

Finalizada a etapa de implementação, realizou-se análise dos casos de teste gerados por meio desta pesquisa. Considerou-se como critério de análise destes casos de teste as características de cobertura exploratória e variação das rotinas de teste.

Realizou-se uma análise dos casos de teste gerados a partir do registro das atividades do usuário da aplicação móvel com intuito de determinar se a pesquisa obteve êxito em alcançar os objetivos propostos. Considerou-se também a documentação gerada a partir dos casos de teste automatizado buscando determinar a conformidade entre estes itens.

É valido ressaltar que durante a etapa de implementação e aplicação desta pesquisa foram realizadas inúmeras execuções da aplicação móvel de teste. Esta alta demanda de execuções, gerou um número considerável de casos de testes por meio dos mais variados cenários.

5.2.1 Cenário de acesso e consulta de registros da aplicação móvel de testes

Para uma análise dos resultados obtidos optou-se pela escolha de um dos cenários executados durante a etapa de implementação desta pesquisa. Optou-se pelo cenário de acesso e consulta de um registro.

Neste cenário onde a aplicação móvel de testes possui uma base de dados com registros persistidos em memória, escolheu-se o item de nome *Unesc* na listagem de itens, na tela de consulta. Realizou-se o acesso de determinado registro por meio da seleção na listagem de itens. Esta atividade resultou na abertura do registro. Após a verificação dos valores em tela ao qual se constatou não ser o registro esperado, utilizou-se do elemento de interface *cancelarBtn* para ter acesso novamente a listagem de itens.

Uma vez nesta listagem, utilizou-se do filtro de busca por meio da entrada de dados no elemento de interface *searchEditTxt*. O dado informado como critério do filtro foi *Unesc*. No retorno dos resultados obtidos verificou-se que apenas um registro possuía o nome desejado. Por último, encerrou-se a execução das atividades e gerou-se o caso de teste automatizado.

Figura 13 – Caso de teste automatizado gerado partir do registro das atividades

```
@Test
public void teste() throws Exception {

    searchEditTxt.clear();
    getElementUsingParentIdAndTextAndScrollTo("list", "Unesc").click();
    Assert.assertEquals(false, elementHasFocus(nomeMotorista));
    Assert.assertEquals("Unesc", nomeMotorista.getText());
    getElementByIdAndScrollTo("nomeMotorista");
    getElementByIdAndScrollTo("cancelarBtn");
    cancelarBtn.click();
    Assert.assertEquals(false, isElementDisplayed(empty_text));
    if(!elementHasFocus(searchEditTxt)){
        searchEditTxt.click();
    }
    Assert.assertEquals(true, elementHasFocus(searchEditTxt));
    searchEditTxt.setValue("Unesc");
    Assert.assertEquals("Unesc", searchEditTxt.getText());
    Assert.assertEquals(false, isElementDisplayed(empty_text));
}
```

Fonte: Do autor.

Na figura 13 pode-se observar o caso de teste automatizado gerado a partir do registro das atividades do usuário durante a etapa de execução do cenário estudado.

A análise do caso de teste gerado indica uma aplicação bem-sucedida da pesquisa mediante ao cenário exposto, uma vez que instruções do algoritmo de teste automatizado estão em conformidade com as ações executadas pelo usuário. Ou seja, a pesquisa foi capaz de assemelhar um caso de teste automatizado ao contexto e forma de utilização da aplicação móvel feita por um usuário.

Contudo, verificou-se que a asserção de quantidade de registros encontrados pelo filtro de busca não está presente no caso de teste automatizado. Isto se dá ao fato da instrução de registro desta atividade não ter sido inserida no código da aplicação de testes.

Entende-se por meio deste resultado que a geração do caso de teste automatizado depende diretamente do uso correto dos métodos de registro de atividade. A omissão de um comando, ou sua utilização de forma incorreta pode refletir na geração de casos de teste que não representam as atividades executadas pelos usuários.

Em paralelo a criação deste caso de teste obteve-se a documentação do teste gerado. Esta documentação especifica os passos executados durante a interação do usuário com a aplicação móvel.

Figura 14 – Documentação do caso de teste do primeiro cenário

```
Documentação do caso de teste
-----------------------------
Passo 1: Limpar o conteúdo do elemento 'searchEditTxt'
Passo 2: Selecionar o item 'Unesc' no elemento 'list'
Passo 3: Verificar se o elemento 'nomeMotorista' está sem o foco
Passo 4: Verificar se o elemento 'nomeMotorista' possui o valor 'Unesc'
Passo 5: Acessar a tela 'Cadastro de Motoristas'
Passo 6: Clicar no elemento 'cancelarBtn'
Passo 7: Acessar a tela 'Listagem de Motoristas'
Passo 8: Verificar se o elemento 'empty_text' está oculto
Passo 9: Focar no elemento 'searchEditTxt'
Passo 10: Verificar se o elemento 'searchEditTxt' está com foco
Passo 11: Digitar o texto 'Unesc' no elemento 'searchEditTxt'
Passo 12: Verificar se o elemento 'searchEditTxt' possui o texto 'Unesc'
Passo 13: Verificar se o elemento 'empty_text' está oculto
```

Fonte: Do autor.

Pode-se considerar válida a documentação ilustrada na figura 14 dado ao fato de que as atividades executadas estão documentadas de forma que possibilite uma possível consulta e reprodução das atividades.

5.2.2 Cenário de edição do cadastro com defeitos de regressão

Em um segundo cenário de testes executado durante a etapa de implementação desta pesquisa executou-se uma rotina especifica de atividades com intuito de demonstrar a capacidade deste estudo de garantir a qualidade de software por meio dos casos de teste gerados.

Neste cenário, realizou-se a inserção de um registro na aplicação móvel de testes. Optou-se por nomear este registro com o nome *Unesc*. Utilizou-se do conteúdo *01234567890* para realizar a entrada de dados no elemento *cpfMotorista*. Realizou-se a persistência do registro por meio do clique no elemento de interface *salvarBtn.* Como comportamento esperado, a aplicação móvel direcionou o usuário para a listagem de registros. Uma vez na listagem de registros, realizou-se a consulta do registro inserido por meio da entrada do dado *Unesc* no elemento de interface *searchEditTxt.* Obteve-se como resultado desta atividade a filtragem da

lista de itens, expondo a presença do registro *Unesc.* Por último, encerrou-se a execução das atividades e gerou-se o caso de teste automatizado.

Figura 15 – Caso de teste automatizado do segundo cenário

```
@Test
public void teste() throws Exception {

    getElementByIdAndScrollTo("add_driver_btn");
    add_driver_btn.click();
    Assert.assertEquals( expected: false, elementHasFocus(nomeMotorista));
    Assert.assertEquals( expected: "", nomeMotorista.getText());
    getElementByIdAndScrollTo("nomeMotorista");
    if(!elementHasFocus(nomeMotorista)){
        nomeMotorista.click();
    }
    Assert.assertEquals( expected: true, elementHasFocus(nomeMotorista));
    nomeMotorista.setValue("Unesc");
    Assert.assertEquals( expected: "Unesc", nomeMotorista.getText());
    if(!elementHasFocus(cpfMotorista)){
        cpfMotorista.click();
    }
    Assert.assertEquals( expected: true, elementHasFocus(cpfMotorista));
    cpfMotorista.setValue("01234567890");
    Assert.assertEquals( expected: "01234567890", cpfMotorista.getText());
    pressKey(AndroidKeyCode.ENTER);
    Assert.assertEquals( expected: false, elementHasFocus(cpfMotorista));
    Assert.assertEquals( expected: "012.345.678-90", cpfMotorista.getText());
    getElementByIdAndScrollTo("salvarBtn");
    salvarBtn.click();
    if(!elementHasFocus(searchEditTxt)){
        searchEditTxt.click();
    }
    Assert.assertEquals( expected: true, elementHasFocus(searchEditTxt));
    searchEditTxt.setValue("Unesc");
    Assert.assertEquals( expected: "Unesc", searchEditTxt.getText());
    Assert.assertEquals( expected: false, isElementDisplayed(empty_text));
}
```

Fonte: Do autor.

Na figura 15 ilustra-se a especificação do segundo caso de teste gerado a partir do registro das atividades do usuário. Nesta variação de caso de teste foram reproduzidas as mesmas atividades realizadas pelo usuário da aplicação móvel. Isto mostra então a realização do objetivo proposto por essa pesquisa de auxiliar no aumento da cobertura de testes automatizado, visto que se possibilitou por meio deste estudo gerar uma variação de caso de teste.

Figura 16 – Documentação do caso de teste do primeiro cenário

```
Documentação do caso de teste
-----------------------------
Passo 1: Acessar a tela 'Listagem de Motoristas'
Passo 2: Clicar no elemento 'add_driver_btn'
Passo 3: Verificar se o elemento 'nomeMotorista' está sem o foco
Passo 4: Verificar se o elemento 'nomeMotorista' está sem nenhum valor
Passo 5: Acessar a tela 'Cadastro de Motoristas'
Passo 6: Focar no elemento 'nomeMotorista'
Passo 7: Verificar se o elemento 'nomeMotorista' está com foco
Passo 8: Digitar o texto 'Unesc' no elemento 'nomeMotorista'
Passo 9: Verificar se o elemento 'nomeMotorista' possui o texto 'Unesc'
Passo 10: Focar no elemento 'cpfMotorista'
Passo 11: Verificar se o elemento 'cpfMotorista' está com foco
Passo 12: Digitar o texto '01234567890' no elemento 'cpfMotorista'
Passo 13: Verificar se o elemento 'cpfMotorista' possui o texto '01234567890'
Passo 14: Tirar o foco do elemento 'cpfMotorista'
Passo 15: Verificar se o elemento 'cpfMotorista' está sem o foco
Passo 16: Verificar se o elemento 'cpfMotorista' possui o valor '012.345.678-90'
Passo 17: Clicar no elemento 'salvarBtn'
Passo 18: Acessar a tela 'Listagem de Motoristas'
Passo 19: Focar no elemento 'searchEditTxt'
Passo 20: Verificar se o elemento 'searchEditTxt' está com foco
Passo 21: Digitar o texto 'Unesc' no elemento 'searchEditTxt'
Passo 22: Verificar se o elemento 'searchEditTxt' possui o texto 'Unesc'
Passo 23: Verificar se o elemento 'empty_text' está oculto
```

Fonte: Do autor.

Por meio da figura 16 pode-se observar a especificação da documentação do caso de teste automatizado. A documentação gerada indica a variação do teste e a cobertura de mais funcionalidades da aplicação móvel durante a execução desta rotina.

Percebe-se por meio dos resultados dos cenários de teste apresentados, que foi possível realizar o registro das atividades do usuário da aplicação móvel. Além disso, este estudo também possibilitou a geração do caso de teste automatizado a partir destas atividades. Da mesma forma, a documentação dos passos executados foi gerada com sucesso em paralelo pelo algoritmo implementado neste estudo.

Em resumo, afirma-se por meio do resultado obtido que se obteve êxito nesta pesquisa em relação ao objetivo proposto de implementar um algoritmo de registro de atividades.

É possível afirmar da mesma forma, que o objetivo de implementar um algoritmo de geração de casos de teste automatizado foi realizado com sucesso.

Mediante as possiblidades de variação de casos de teste por meio da execução desta pesquisa, pode-se afirmar que este estudo possibilitou uma

melhoria na criação de casos de teste automatizado, pois tornou prático o processo de especificação de casos de teste exploratórios com riqueza de detalhes e documentação devida.

5.2.3 Considerações finais

Como visto anteriormente na fundamentação teórica, o processo de garantia de qualidade de software é fundamental para os sistemas de software. Quando agregado ao processo de automação de software eleva as chances de detecção de não conformidades no sistema.

Em contrapartida, casos de teste automatizado de baixa qualidade possuem características que não condizem com o contexto de uso dos usuários, o que leva muitas vezes a uma prática ineficiente do processo de automação de software. Além disso, casos de teste saturados, que exercitam exaustivamente as mesmas rotinas possuem menos chances de detecção de defeitos, e tendem a permitir que esses defeitos sejam liberados para o mercado.

Sabe-se que um dos motivos para que isso aconteça está relacionado ao fato de que o processo de criação de casos de teste automatizado é custoso, e exige um tempo elevado para sua especificação. Quando aliado ao fato do responsável pela especificação dos casos de teste não possuir pleno domínio dos requisitos do sistema e conhecimento de linguagens de programação o cenário tende a piorar (PFALLER et al., 2008, tradução nossa).

Isto, pode se tornar um desafio considerando o perfil dos profissionais atuantes na área de teste de software refletindo na qualidade dos casos de teste automatizado. Pode-se dizer que a qualidade de um caso de teste é medida pela cobertura dos testes especificados e a capacidade de reproduzir rotinas com características exploratórias (MORITZ, 2009, tradução nossa).

Mediante aos problemas constatados, apresenta-se os resultados obtidos por meio da aplicação desta pesquisa. Constatou-se por meio dos resultados obtidos que a aplicação desta pesquisa oferece uma alternativa para minimizar os problemas apresentados em relação a falta de testes exploratórios e a baixa qualidade de cobertura de testes.

Afirma-se por meio da execução deste estudo que se obteve uma melhoria na criação dos casos de teste automatizado, alcançando os resultados esperados.

Determinou-se a execução desta pesquisa como bem-sucedida por meio do cumprimento de requisitos qualitativos referentes aos casos de teste automatizado gerados. A partir da análise realizada constatou-se as características que indicam a melhoria na criação de casos de teste automatizado (GARRETT, 2012, tradução nossa):

a) capacidade de reproduzir as atividades dos usuários por meio de teste automatizado;
b) fidelidade entre a especificação dos casos de teste automatizado com o contexto de utilização dos usuários;
c) aumento da cobertura dos casos de teste automatizado e aumento da capacidade de revelar defeitos;
d) aprimoramento da característica exploratória dos casos de teste automatizado indicando diversificação dos passos contidos nos casos de teste automatizado.

Observa-se por meio da análise deste caso especifico, que não se descarta a possibilidade de aplicação deste estudo em outras aplicações móveis. Acredita-se que a execução bem-sucedida desta pesquisa em outras aplicações móveis poderá proporcionar os mesmos benefícios resultantes neste estudo.

Constatou-se também, por meio dos resultados obtidos, que a eficiência desta pesquisa em gerar casos de teste automatizado e ampliar a cobertura de testes exploratórios tende a crescer a medida que o número de usuários da aplicação móvel aumenta. Isto ocorre dado ao fato de que as atividades executadas por cada usuário diferem entre si. Consequentemente há um aumento na diversidade dos cenários e das atividades utilizadas como base para geração de novos casos de teste.

Observou-se durante os estudos iniciais sobre algoritmos de teste automatizado que a especificação manual de casos de teste automatizado exige um nível de esforço elevado. Isto foi possível por meio da comparação do tempo investido para criação dos de teste automatizado e utilização da aplicação deste estudo. Constatou-se uma redução considerável no tempo necessário para especificação de casos de teste automatizado. Esta diminuição no tempo investido

para especificação dos casos de teste automatizado possibilita a equipe de testes realizar aprimoramentos no processo de garantia de qualidade de software.

Com relação a singularidade desta pesquisa em comparação com as bibliografias consultadas, e estudos realizados anteriormente a este não foram encontradas referenciais teóricos que se assemelhem ao que foi obtido por meio desta pesquisa. Constatou-se por meio de consulta ao trabalho abordados no capítulo 4 a existência de estudos sobre geração de casos de teste automatizado, contudo, estes estudos abordam diferentes estratégias de teste e problemas a serem resolvidos.

6 CONCLUSÃO

O desenvolvimento e a aplicação dos algoritmos de registro de atividades e geração de caso de teste automatizado proporcionaram a experimentação de recursos e estratégias que auxiliam o processo de especificação do teste automatizado.

A elaboração deste estudo proporcionou conhecimento sobre os desafios existentes da adoção do processo de testes em aplicações móveis. Por meio do referencial teórico foram abordados os fatores fundamentais que devem ser considerados na implantação de uma estratégia de automação de testes.

6.1. PRINCIPAL CONTRIBUIÇÃO DA PROPOSTA

A aplicação desta pesquisa se mostrou de grande contribuição para o processo de garantia de qualidade de software por meio a geração de casos de teste automatizado como um processo paralelo à utilização de aplicações móveis. Percebe-se por meio dos resultados obtidos que este processo possibilitou a criação de casos de teste exploratórios em grande escala, aumentando consideravelmente a quantidade de testes exploratórios.

Conclui-se também, por meio dos da análise dos resultados, que houve contribuição na diminuição do tempo necessário para especificação de casos de teste automatizado.

Pode-se também considerar como uma vantagem desta pesquisa a possiblidade de participação de equipes externas na criação de casos de teste automatizado, incluindo usuários participantes de testes de aceitação, tratado na subseção 2.2.3, usuários finais em ambiente de produção e equipes de testes de software. Isto se dá ao fato da geração dos casos de teste ocorrer de forma transparente para os usuários e pode ser realizada por meio da própria aplicação móvel.

Conclui-se por meio da aplicação desta pesquisa, que a cobertura e a qualidade dos casos de teste exploratórios depende de diversos fatores, que variam desde a experiência do testador em especificação de casos de teste até o conhecimento sobre as regras de negócio da aplicação. Estes fatores possuem influência direta na quantidade e na qualidade dos testes exploratórios.

6.2 LIMITAÇÕES DA PROPOSTA

Durante a aplicação e execução deste estudo, foram vistas dificuldades em determinar comportamentos corretos mediante a falhas na execução dos casos de testes. Gerou-se por meio deste estudo inúmeros casos de teste utilizando os registros de atividades do usuário da aplicação móvel, porém em alguns casos não se pode determinar se os casos de teste apenas replicavam um comportamento incorreto da aplicação móvel de testes. Em um ambiente de produção, é fundamental que estas falhas sejam diferenciadas de defeitos reais do sistema para que não sejam interpretados como falsos positivos.

Outra dificuldade a qual deparou-se no decorrer da aplicação desta pesquisa, foi fato da especificação dos casos de teste automatizado ser realizada de forma automática dificultando o trabalho do testador de distinguir casos de teste repetidos, isto gera uma incerteza para a equipe de testes, pois dificulta a etapa de decisão da relevância de determinados testes.

Como forma de amenizar este problema, propõem-se no decorrer deste capítulo algumas ideias para o desenvolvimento de melhorias que permitem aprimorar os resultados obtidos por meio da aplicação desta pesquisa.

6.3 MELHORIAS FUTURAS

A aplicação do conteúdo abordado a respeito do processo de automação de testes possibilita a variação entre linhas de estudo. Sugere-se como melhoria para este estudo a abordagem das seguintes pesquisas:

a) estudo a respeito da comparação dos casos de teste gerados de forma automatizada com objetivo de determinar o nível de semelhança e relevância entre casos de teste;
b) estudo de um algoritmo automatizado capaz de realizar a análise de casos de teste automatizado distinguindo comportamentos incorretos com base em documentos de requisitos;
c) estudo a respeito da aplicação desta pesquisa em outras plataformas e sistemas operacionais, incluindo IOS.

REFERÊNCIAS

ABDULRAZEG, Ala. A.; NORWAWI, Norita Md, BASIR, Nurlida. **Extending V-model practices to support SRE to build secure web applications** pelo. In: International Conference on Advanced Computer Science and Information System, Jakarta, 2014. Disponível em: <https://ieeexplore.ieee.org/abstract/document/7065838/>. Acesso em: 07 maio 2018.

AMALFITANO, Domenico; FASOLINO, Anna Rita; TRAMONTANA, Porfirio. **A GUI Crawling-Based Technique for Android Mobile Application Testing** pelo. In: 2011 IEEE Fourth International Conference on Software Testing, Verification and Validation Workshops. Disponível em: <https://ieeexplore.ieee.org/document/5954416/>. Acesso em: 07 setembro 2017.

ANDROID. **UiSelector.** Disponível em: <https://developer.android.com/reference/android/support/test/uiautomator/UiSelector>. Acesso em: 03 jun. 2018.

APPIUM. **Introduction to Appium:** Introduction to Appium's Philosophy, Design and Concepts. 2017. Disponível em: <http://appium.io/introduction.html>. Acesso em: 28 nov. 2017.

BACH, James. ***Exploratory Testing Explained***. 2003. Disponível em: <http://www.satisfice-.com/articles/et-article.pdf>. Acesso em: 27 nov. 2017.

BARTIÉ, Alexandre. **Garantia da Qualidade de Software.** Rio de Janeiro: Elsevier Editora Ltda., 2002. Disponível em: <https://books.google.com.br/books?id=O57Us2kUh4oC&printsec=frontcover&hl=pt-BR#v=onepage&q&f=false>. Acesso em: 25 set. 2017.

BASTOS, Anderson et al. **Base de conhecimento em teste de Software.** São Paulo: Martins, 2007. p.35-70

BERNARDO, Paulo Cheque; KON, Fabio. **A Importância dos Testes Automatizado**s. Artigo publicado na Engenharia de Software Magazine, 2008. Disponível em: <https://www.ime.usp.br/~kon/papers/EngSoftMagazine-IntroducaoTestes.pdf >. Acesso em: 2 nov. 2017.

BOUCHER, Mathieu; MUSSBACHER, Gunter. **Transforming Workflow Models into Automated End-to-End Acceptance Test Cases**, 2017 IEEE/ACM 9th International Workshop on Modelling in Software Engineering (MiSE), Buenos Aires, 201. Disponível em: <https://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=7964598>. Acesso em: 01 maio 2017.

CALABASH. **Introduction to Calabash.** Disponível em: <https://developer.xamarin.com/guides/testcloud/calabash/introduction-to-calabash/>. Acesso em: 27 nov. 2017.

CRISPIN, Lisa; GREGORY, Janet. **Agile Testing: A Practical Guide for Testers and Agile Teams**. Pearson Education, 2008. ISBN 9780321616937. Disponível em: < https://books.google.com.br/books?id=68_IhPvoKS8C >. Acesso em: 16 agosto 2017.

DUSTIN, Elfriede; RASHKA, Jeff; PAUL, John. **Automated Software Testing: Introduction, Management, and Performance.** Pearson Education, 1999. 608 p. Disponível em: <https://books.google.com.br/books?id=kl2H0G6EFf0C>. Acesso em: 28 maio 2017

FERRARI, Fabricio et al. **Visões sobre Teste de Software:** Diferentes perspectivas da comunidade de teste brasileira discutidas no DFTestes. 2009. Disponível em: <https://s3.amazonaws.com/beacon.cnd/977fcb1d80b47834.pdf?t=1482409721>. Acesso em: 26 nov. 2017.

FEWSTER, Mark; GRAHAM, Dorothy. **Software Test Automation: Effective use of test automation tools.** Great Britain: ACM Press Books, 1999. Disponível em: <https://books.google.com.br/books/about/Software_Test_Automation.html?id=z2QABZKTihQC&redir_esc=y>. Acesso em: 16 agosto 2017.

GAO, Jerry et al. Mobile Application Testing: A Tutorial. **Computer**, [s.l.], v. 47, n. 2, p.46-55, fev. 2014. Institute of Electrical and Electronics Engineers (IEEE). http://dx.doi.org/10.1109/mc.2013.445. Disponível em: <https://ieeexplore.ieee.org/document/6693676/>. Acesso em: 27 maio 2018.

GARRETT, Thom. **Useful Automated Software Testing Metrics.** 2012. Disponível em: <http://idtus.com/wp-content/uploads/2012/09/UsefulAutomatedTestingMetrics.pdf>. Acesso em: 03 maio 2018.

GEBIZLI, Ceren Sahin; SOZER, Hasan. Impact of Education and Experience Level on the Effectiveness of Exploratory Testing: An Industrial Case Study. **2017 Ieee International Conference On Software Testing, Verification And Validation Workshops (icstw)**, [s.l.], p.23-28, mar. 2017. IEEE. http://dx.doi.org/10.1109/icstw.2017.8. Disponível em: <https://ieeexplore.ieee.org/document/7899025/>. Acesso em: 27 maio 2018.

KNOTT, Daniel. **Hands-On Mobile App Testing: A Guide For Mobile Testers and Anyone Involved in the Mobile App Business.** Indiana: Addison-wesley - Professional, 2015. 230 p. Disponível em: <http://ptgmedia.pearsoncmg.com/images/9780134191713/samplepages/9780134191713.pdf>. Acesso em: 25 fevereiro 2018.

KOCHHAR, Pavneet Singh et al. Understanding the Test Automation Culture of App Developers. **2015 Ieee 8th International Conference On Software Testing, Verification And Validation (icst)**, [s.l.], p.25-35, abr. 2015. IEEE. Disponível em: <https://ieeexplore.ieee.org/document/7102609/>. Acesso em: 01 jul. 2017.

LEUNG, R. P. H. Karl; YEUNG, Wing Lok. **Generating User Acceptance Test Plans from Test Cases** pelo. In: 31st Annual International Computer Software and Applications Conference (COMPSAC 2007), 31., Beijing, 2007. Disponível em: <https://ieeexplore.ieee.org/document/1204375/>. Acesso em: 07 setembro 2017.

MALL, Rajib. **Fundamentals Of Software Engineering**, PHI Learning Pvt. Ltd, 2009. Disponível em: <https://www.researchgate.net/publication/220689590_Fundamentals_of_software_engineering_2_ed>. Acesso em: 04 maio 2017.

MIGUEL, Sérgio Barriviera. **Padrão para Documentação de Teste de Software**. 2015. Disponível em: <http://www.devmedia.com.br/padrao-para-documentacao-de-teste-desoftware/26534>. Acesso em: 27 nov. 2017.

MOLINARI, Leonardo. **Inovação e Automação de Testes de Software.** São Paulo: Erica, 2010.p.25-40.

MORITZ, Evelyn. **Case study: How analysis of customer found defects can be used by system test to improve quality**. In: INTERNATIONAL CONFERENCE ON SOFTWARE ENGINEERING - COMPANION VOLUME, 31**.** Vancouver: Ieee, 2009. p. 123 - 129. Disponível em: <https://ieeexplore.ieee.org/document/5070970/>. Acesso em: 04 jun. 2017.

MYERS, Glenford J.; SANDLER, Corey; BADGETT, Tom. **The Art of Software Testing**. Wiley, 2011. Disponível em: <https://books.google.com.br/books?id=GjyEFPkMCwcC>. Acesso em: 07 agosto 2017.

NAIK, Kshirasagar; TRIPATHY, Priyadarshi. **Software Testing and Quality Assurance.** Hoboken (New Jersey): John Wiley & Sons, Inc., 2008.
NOGUEIRA, Elias. **INTRODUÇÃO AO ROBOTIUM.** 2017. Disponível em: <http://www.qualister.com.br/blog/introducao-ao-robotium>. Acesso em: 28 nov. 2017.

PFAHL, Dietmar et al. **How is exploratory testing used?** Proceedings Of The 8th Acm/ieee International Symposium On Empirical Software Engineering And Measurement - Esem '14, [s.l.], p.1-2, 2014. ACM Press. Disponível em: <https://dl.acm.org/citation.cfm?id=2652531>. Acesso em: 02 maio 2017.

PFALLER, Christian et al. Multi-Dimensional Measures for Test Case Quality. **2008 Ieee International Conference On Software Testing Verification And Validation Workshop**, [s.l.], p.1-4, 2008. IEEE. http://dx.doi.org/10.1109/icstw.2008.28. Disponível em: <https://ieeexplore.ieee.org/document/4567035/>. Acesso em: 07 jun. 2018.

PRESSMAN, Roger S.; MAXIM, Bruce R. **Engenharia de Software:** Uma Abordagem Profissional. 8. ed. Nova Iorque: Mariana Belloli, 2016. 968 p.

RAAPPANA, Paula et al. The Effect of Team Exploratory Testing -- Experience Report from F-Secure. **2016 Ieee Ninth International Conference On Software**

Testing, Verification And Validation Workshops (icstw), [s.l.], p.295-304, abr. 2016. IEEE. http://dx.doi.org/10.1109/icstw.2016.13. Disponível em: <https://ieeexplore.ieee.org/document/7528976/>. Acesso em: 27 maio 2018.

ROCHA, A. R. C., MALDONADO, J. C., WEBER, K. C. **Qualidade de software - Teoria e prática**, Prentice Hall, São Paulo, 2001.p.25-70. Disponível em: <https://books.google.com.br/books/about/Qualidade_de_software.html?hl=pt-BR&id=gBtGAAAAYAAJ&redir_esc=y>. Acesso em: 04 maio 2017.

SILVA, Paulo César Barreto da; ALVES, Thiago Salhab; BRUNO, Elisângela Andrade. Automação De Testes Funcionais: Testes funcionais automatzados de sofwar. **Revista de Ciências Exatas e Tecnologia,** Santa Barbara, v. 6, n. 6, p.113-133, 28 abr. 2014.

SOASTA, **Five Strategies for Performance Testing Mobile Application**s, a white paper, 2011. Disponível em: < http://info.soasta.com/performance-testing-mobile-apps-sem2.html>. Acesso em: 25 nov. 2017.

SOMMERVILLE, Ian. **Engenharia de software**. PEARSON BRASIL, 2011. ISBN 9788579361081. Disponível em: <https://books.google.com.br/books?id=H4u5ygAACAAJ >. Acesso em: 07 maio 2017.

STANDISH GROUP. **The Standish Group Report:** chaos. 2014. Disponível em: <https://www.projectsmart.co.uk/white-papers/chaos-report.pdf>. Acesso em: 07 jul. 2018.

TANNOURI, Patricia Aline. **O que é Testabilidade**? Disponível em: <http://www.linhadecodigo.com.br/artigo/923/o-que-e-testabilidade.aspx>. Acesso em: 26 nov. 2017

VARHOL, Peter. **Why Exploratory Testing Should Be Part of Your Process.** Disponível em: <https://www.telerik.com/blogs/why-exploratory-testing-should-be-part-of-your-process>. Acesso em: 31 maio 2018.

WEISS, Johannes; SCHILL, Alexander. **Specifying Executable or Nonexecutable Acceptance Test Cases for Event Processing Applications** pelo. In: 42th Euromicro Conference on Software Engineering and Advanced Applications (SEAA), Limassol, 2016. Disponível em: <https://ieeexplore.ieee.org/document/7592802/>. Acesso em: 07 agosto 2017.

WISSINK, Tom; AMARO, Carlos. **Successful Test Automation for Software Maintenance.** pelo. In: IEEE International Conference on Software Maintenance, 2006. Disponível em: <https://ieeexplore.ieee.org/document/4021345/>. Acesso em: 07 maio 2018

XAMARIN. **Introduction to Xamarin Test Cloud.** Disponível em: <https://developer.xamarin.com/guides/testcloud/introduction-to-test-cloud/>. Acesso em: 27 nov. 2017.

APÊNDICE(S)

Apêndice A – Artigo

Proposta Para Melhoria Na Criação De Testes Automatizados Utilizando Os Registros Das Atividades Dos Usuários De Aplicações Móveis

Fabricio Medeiros Somini[1], Ana Claudia Garcia Barbosa[1]

[1]Univesidade do Extremo Sul Catarinense (UNESC) – Criciúma, SC – Brasil

fabricio.somini@gmail.com, agb@unesc.net

Abstract. *Software development companies in order to meet the software market requirements have been using the software testing process as a way to ensure product quality. Considering this scenario, in this research, a procedure was developed that consists of a library for the creation of automated tests with an exploratory aspect for mobile applications. Through the utilization of this study, the creation of automated test cases was performed based on the records of the mobile user activities. Among the performed tests and the analysis of the generated test cases, an improvement in the quality of the automated test cases with an exploratory aspect has been confirmed.*

Resumo. *As empresas de desenvolvimento de software com intuito de cumprir as exigências do mercado utilizam o processo de teste de software como forma de garantir a qualidade do produto. Tendo em vista esse cenário, nesta pesquisa foi desenvolvido um procedimento que consiste em uma biblioteca de criação de testes automatizados de aspecto exploratório para aplicações móveis. Por meio da aplicação deste estudo realizou-se a criação de casos de teste automatizado com base no registro das atividades dos usuários de aplicações móveis. Dentre os testes realizados e diante da análise dos casos de teste gerados confirmou-se uma melhoria na qualidade dos casos de teste automatizado com aspecto exploratório.*

1. Introdução

O processo de testes nem sempre foi priorizado como uma atividade no processo de desenvolvimento. Em muitos casos, se o prazo para entrega do sistema não fosse o suficiente as empresas desconsideravam a execução das atividades do teste de software. Contudo, este comportamento vem se tornando cada vez menos frequente, dado ao fato de muitos defeitos serem encontrados em sistemas já liberados para os clientes, diminuindo a credibilidade do produto no mercado de trabalho, assim como a confiança dos clientes neste produto (SOMMERVILLE, 2011).

O teste de software é uma atividade eficaz para a garantia de qualidade de software, porém a implementação é um grande desafio devido à alta complexidade dos sistemas e às inúmeras dificuldades relacionadas às etapas de desenvolvimento. Outro fator que torna o processo de testes uma atividade desafiadora é a necessidade de repetir os casos de teste durante o processo de desenvolvimento do software levando muitas empresas a optarem pela automação dos casos de teste (BASTOS et al., 2007; BERNARDO, 2011).

Os casos de teste automatizados com características de teste exploratório são complexos e rico em detalhes, por este motivo possuem alto custo de criação e manutenção. Desta forma, costuma-se observar em empresas de desenvolvimento de software que muitos casos de teste não representam o contexto real da forma de utilização dos usuários (BARTIÉ, 2002).

Analogamente, os usuários exercem diferentes atividades durante a utilização do sistema. Estas atividades são variadas e ocorrem em diversos cenários que dificilmente serão previstos pelos responsáveis pela especificação dos testes automatizados devido às inúmeras possibilidades de caminhos a serem seguidos. Como consequência, muitos casos de teste não preveem comportamentos e ações inesperadas dos usuários (MORITZ, 2009, tradução nossa).

Quando não previstas pela equipe de qualidade, estas atividades permanecem sem a cobertura de testes exploratórios devida, expondo o sistema a cenários de vulnerabilidade em que a presença de defeitos pode ser revelada durante a sua utilização (PRESSMAN; MAXIM, 2016).

Sendo assim propõe-se por meio desta pesquisa um procedimento que consiste no desenvolvimento de uma biblioteca de criação de testes automatizados para aplicações móveis buscando minimizar o problema da falta de casos de teste exploratório.

2. Proposta Para Melhoria Na Criação De Testes Automatizados Utilizando Os Registros Das Atividades Dos Usuários De Aplicações Móveis

Esta pesquisa é um procedimento que consiste no desenvolvimento de uma biblioteca de criação de testes automatizados para aplicações móveis. Propõe-se por meio desta pesquisa uma melhoria na criação de casos de teste automatizado minimizando o problema da falta de cobertura de testes exploratórios para aplicações móveis.

Pretende-se alcançar essa melhoria por meio da disponibilização de recursos e métodos de registro de atividades desenvolvidos nesta pesquisa. Utilizando-se da biblioteca de criação de testes busca-se possibilitar a geração de novos casos de teste automatizado utilizando os registros das atividades. Além disso, busca-se possibilitar que as equipes responsáveis pela qualidade do teste de software tenham acesso a este recurso auxiliando na etapa de especificação de casos de teste automatizado.

No desenvolvimento deste trabalho foram empregados algoritmos de teste automatizado para aplicações móveis. Foram gerados casos de teste automatizado a partir de algoritmos de teste automatizado. Realizou-se a implementação de uma aplicação móvel de testes com intuito de empregar os algoritmos de teste automatizado.

Os resultados deste estudo foram obtidos com base na análise feita a partir dos casos de teste gerados considerando a sua capacidade de reproduzir as atividades dos usuários e realizar testes exploratórios.

2.1 Investigação dos algoritmos de teste automatizado para aplicações móveis

Durante a elaboração desta pesquisa foram realizados estudos sobre algoritmos de teste automatizado para aplicações móveis. Compreendeu-se que algoritmos de teste automatizado possibilitam a reprodução de determinadas atividades de interação com aplicações móveis. As atividades são reproduzidas por meio execução de comandos de programação. Estes comandos representam atividades de uma aplicação móvel incluindo cliques e o pressionamento de teclas.

2.2 Algoritmo de registro de atividades

Nesta etapa iniciou-se o desenvolvimento do algoritmo de registro de atividades de usuários de aplicações móveis.

O objetivo deste algoritmo é registrar as atividades que dificilmente são consideradas durante a etapa de especificação de casos de teste automatizado. Optou-se por utilizar estas atividades como base para melhoria da criação de casos de teste exploratório.

Para delimitar um escopo de estudo optou-se por realizar o registro das seguintes atividades executadas em aplicações móveis: navegação entre as telas, entrada de valores em campos, foco em campos, pressionamento de teclas, seleção de opções em caixas de opções, seleção de itens em listagens, alteração de estados de caixas de seleção, movimentação de barras de progresso e cliques em botões.

A escolha destas atividades se mostrou satisfatória para alcançar o objetivo proposto pelo algoritmo de registro de atividades, pois envolvem diferentes tipos de rotinas aumentando a diversidade da forma de execução para cada uma destas atividades.

O algoritmo de registro de atividades tem sua especificação contida em uma classe denominada *RegistroAtividades*. Na tabela 1 têm-se os métodos implementados relativos ao registro das atividades:

Função	Tipo	Descrição
inicializar	void	Inicializa as variáveis
vincularElemento	Atividade	Vincula o elemento de interface a uma instancia da classe *Atividade*
registrarAcessoTela	void	Registra o nome da tela acessada pelo usuário
pressionar	void	Simula o pressionamento de uma tecla física do aparelho móvel
adicionarAtividade	void	Adiciona uma atividade na lista de atividades registradas
getListaAtividades	ArrayList<Atividade>	Retorna a lista de atividades registradas
getINSTANCE	RegistroAtividades	Retorna uma instancia da classe *RegistroAtividades*

Tabela 1 – Métodos da Classe *RegistroAtividades*

Considerando o método *vincularElemento* apresentado na tabela 1 que realiza a criação do vínculo entre o elemento de interface e uma instância da classe *Atividade*.

Os métodos do algoritmo de registro de atividades realizam o armazenamento do tipo da atividade e das informações necessárias para sua reprodução. Quando atividade é encerrada o método *reproduzirAcoes* adiciona esta atividade a lista de atividades registradas.

A lista das atividades registradas durante a execução do algoritmo de registro de atividades pode ser obtida por meio do método *getListaAtividades*.

A listagem é utilizada pelo algoritmo de geração de casos de teste automatizado para a criação de novos casos de teste automatizado.

A utilização do algoritmo de registro de atividades busca contribuir com processo de teste de software por meio do aumento da capacidade exploratória dos casos de teste automatizado e da diversidade na forma de especificação dos casos de teste.

2.3 Algoritmo de geração de casos de teste automatizado

Nesta etapa, iniciou-se o desenvolvimento do algoritmo de geração de casos de teste automatizado. Propõem-se por meio deste algoritmo uma melhoria na criação de casos de

teste automatizado, utilizando-se das atividades registradas pelo algoritmo de registro de atividades. Esta abordagem busca ampliar a cobertura de casos de teste exploratórios aumentando a semelhança com o contexto de uso dos usuários.

O algoritmo de geração de casos de teste automatizado foi especificado por meio da classe *GeradorCasosTeste*. Neste algoritmo estão dispostos métodos que permitem a criação de um caso de teste a partir de registros de atividades. A tabela 2 demonstra estes métodos:

Função	Tipo	Descrição
inicializar	void	Inicializa as variáveis
gerar	Teste	Gera o caso de teste

Tabela 2 – Métodos da classe *GeradorCasosTeste*

Por meio da tabela 2 pode-se observar o método *inicializar* que recebe como parâmetro uma variável do tipo *Setup* onde são especificadas as configurações para execução do caso de teste automatizado. O segundo parâmetro deste método permite o envio de uma variável do tipo *Preferencias* que define as características de especificação do caso de teste automatizado. Por meio da especificação deste parâmetro é possível definir as seguintes características: pacotes de importação, nome do pacote de testes, nome da classe pai, declaração de métodos específicos, declaração do método *tearDown*. As variáveis *setup* e *preferencias* são fundamentais para o funcionamento do método *gerar*.

O método *gerar* recebe como parâmetro uma lista de objetos do tipo *Atividade*. A lista de atividades é convertida em um algoritmo de teste automatizado. O segundo parâmetro do método *gerar* é dado pelo nome dado ao caso de teste.

O resultado da execução do método *gerar* é um objeto do tipo *Teste* que contém o caso de teste automatizado e a documentação dos passos reproduzidos pelo teste.

A utilização do algoritmo de geração de casos de teste automatizado busca possibilitar ao processo de teste de software alcançar um nível maior de conformidade entre os casos de teste e a documentação especificada. Isto contribui com o processo de garantia de qualidade de software, e aumenta a credibilidade e confiabilidade do sistema.

2.4 Aplicação do conteúdo desenvolvido

Os algoritmos de registro de atividades e geração de casos de teste automatizado foram aplicados em uma aplicação móvel de testes durante o desenvolvimento deste estudo. A aplicação móvel foi desenvolvida para que se pudesse realizar a simulação de um usuário durante a atividade de interação enquanto realiza-se o registro das atividades e geração de casos de teste automatizado.

A aplicação móvel de testes consiste em um sistema que permite o cadastro e consulta de registros de motoristas. Criou-se uma lista de requisitos de interface as quais deseja-se atender por meio do desenvolvimento da aplicação móvel de testes. Busca-se por meio destes requisitos possibilitar a aplicação dos algoritmos de registro de atividade e geração de casos de teste automatizado de forma devida.

Pode-se citar os seguintes requisitos que a aplicação móvel de testes atende: navegação das telas, elementos de interface com conteúdo editável, botões de ação, listagem de itens, barras de progresso, caixas de seleção e caixas de opções. Realizou-se a implementação de duas telas possibilitando ao usuário navegar na aplicação ampliando as possibilidades de variações das atividades executadas.

Os algoritmos de registro de atividades e geração de casos de teste automatizado foram armazenados em uma biblioteca do tipo *AAR*. Esta biblioteca foi publicada no repositório público *JCenter* possibilizando a sua utilização como uma dependência de

projetos de aplicações móveis. A importação da biblioteca de testes foi realizada a partir da inserção da dependência: *implementation 'appiumrunner:unesctcc3:6.25.21.32'*

Após a importação das dependências no projeto, é possível realizar a aplicação do algoritmo de registro de atividades. A aplicação do código do algoritmo de registro de atividades é feita por meio da inserção de uma instrução no código da aplicação móvel de testes. Para isto, mapeiam-se os eventos nativos da aplicação móvel de testes onde deseja-se realizar a inserção da instrução.

Neste estudo foram inseridas instruções em eventos nativos da aplicação móvel de testes buscando abranger todas as áreas do sistema. A especificação das instruções de registro de atividades é fundamental para que as atividades sejam registradas no momento devido.

Especifica-se em um primeiro momento a instrução que indica o início do processo de registro das atividades. Esta instrução consiste na especificação do método *inicializar*.

Em seguida, realiza-se a chamada do método *vincularElemento* indicando o elemento de interface ao deve-se realizar o vínculo do algoritmo de registro de atividades.

A especificação deste método é realizada de forma encadeada, onde o resultado de uma chamada retorna o mesmo tipo de objeto *Atividade*. Isto possibilita que para o mesmo elemento a chamada de registro de atividades seja feita apenas uma vez.

A chamada dos métodos de registro de atividade *focarCampo* encadeando o método *escreverValor* pode ser realizada para registar as atividades de foco e entrada de valores no elemento de interface *nomeMotorista* localizado na tela de cadastro. No final do método de registro de atividade realiza-se o a especificação do método *reproduzirAcoes* que adiciona as atividades registradas à lista de atividades. E o método *verificarValores* adiciona uma atividade assertiva a lista de atividades.

O encerramento do registro das atividades é realizado por meio da especificação do método *gerar* da classe *GeradorCasosTeste*. Este método recebe as atividades registradas realizando a criação de um caso de teste automatizado e gerando a documentação para os testes criados.

Para obter o caso de teste e a documentação dos testes gerados deve-se armazenar o retorno do método *gerar* em uma variável do tipo *Teste*. É de responsabilidade da equipe de testes gerar o arquivo de teste com extensão *.java* onde será inserido o caso de teste.

2.5 Aplicação prática da pesquisa em ambientes de teste

A aplicação desta pesquisa em um ambiente de teste proporcionou uma perspectiva dos possíveis desafios e as vantagens consequentes da utilização deste procedimento.

Na etapa de procedimentos iniciais realizou-se uma análise da aplicação móvel de testes com intuito determinar a sua maturidade mediante a aplicação de uma abordagem de testes de sistema. Determinou-se a utilização da técnica de teste caixa preta com foco no teste de regressão. Buscou-se determinar a viabilidade da implantação do processo de automação, abordando as precauções que devem ser tomadas ao considerar a implantação de testes automatizados.

Definiu-se quais funcionalidades da aplicação móvel de testes deseja-se pela aplicação dos algoritmos de registro de atividades e geração de casos de teste automatizado. Por fim, deu-se início ao planejamento dos casos de teste.

O planejamento da aplicação dos algoritmos de registro de atividades e geração de casos de teste automatizado foi feito com base em uma estratégia de testes exploratórios. Buscou-se desta forma a abranger o maior número de áreas possíveis dentro da aplicação móvel de testes.

Para determinar os pontos de aplicação dos algoritmos, considerou-se a maturidade de cada funcionalidade, e a probabilidade de revelar defeitos no sistema. Foram atribuídas as

responsabilidades ao usuário de utilizar a aplicação durante o processo de registro das atividades. Busca-se com esta atividade realizar a execução da aplicação móvel de testes com base na experiência adquirida por meio de experiências e resultados de baterias de teste anteriores.

A etapa de preparação foi realizada em paralelo com as demais etapas do ciclo de vida do teste de software com objetivo de agilizar o processo de preparação. Nesta etapa realizou-se a verificação dos recursos necessários a execução dos testes. Realizou-se a configuração do emulador de dispositivos móveis para plataforma Android, efetuou-se a instalação e configuração da aplicação móvel de testes no emulador, fez-se a configuração da ferramenta de automação Appium e da IDE Android Studio.

Na etapa de especificação realizou-se a aplicação dos algoritmos de registro de atividades e geração de casos de teste automatizado na aplicação móvel de testes. Neste momento, realizou-se a aplicação dos métodos de registro de atividades nos eventos de elementos de interface.

Na etapa de execução da aplicação móvel de testes buscou-se realizar a execução das atividades de forma coerente, seguindo fluxos lógicos. Optou-se por rotinas com características de execução de testes de sistema, e garantia das conformidades realizadas em testes de usabilidade. Em um segundo momento, foram realizadas variações de aspecto exploratório. Executou-se diversas vezes as mesmas atividades intercalando entre comportamentos e telas do sistema e formas de execução.

Nesta etapa finalizou-se a bateria de testes com a entrega do caso de teste automatizado e a documentação dos testes. Esta etapa foi documentada por meio do armazenamento do teste gerado em um arquivo de extensão *.java* e a documentação do teste foi armazenada em um arquivo de texto. Espera-se que o conteúdo obtido por meio da execução dos testes indique a reprodução das rotinas realizadas pelo usuário.

Tendo concluído a execução das etapas do ciclo de vida do teste de software, iniciou-se uma análise do caso de teste gerado a fim de determinar a sua cobertura exploratória. Realizou-se a execução do caso de teste para garantir que as atividades executadas pelo usuário foram replicadas para o caso de teste automatizado. Utilizou-se dos resultados dos testes para determinar a conformidade entre as atividades executadas, o caso de teste automatizado. A documentação gerada a partir do algoritmo de geração de casos de teste automatizado também foi considerada durante esta análise.

Os resultados obtidos durante a aplicação e execução desta pesquisa serão discutidos no decorrer deste texto com intuito de determinar o êxito na obtenção dos resultados esperados e dos objetivos propostos por meio deste estudo.

3. Resultados

Finalizada a etapa de implementação, realizou-se análise dos casos de teste gerados por meio desta pesquisa. Considerou-se como critério de análise destes casos de teste as características de cobertura exploratória e variação das rotinas de teste.

Realizou-se uma análise dos casos de teste gerados a partir do registro das atividades do usuário da aplicação móvel com intuito de determinar se a pesquisa obteve êxito em alcançar os objetivos propostos. Considerou-se também a documentação gerada a partir dos casos de teste automatizado buscando determinar a conformidade entre estes itens.

Para uma análise dos resultados obtidos optou-se pela escolha de um dos cenários executados durante a etapa de implementação desta pesquisa. Optou-se pelo cenário de acesso e consulta de um registro.

Neste cenário onde a aplicação móvel de testes possui uma base de dados com registros persistidos em memória, escolheu-se o item de nome *Unesc* na listagem de itens, na

tela de consulta. Realizou-se o acesso de determinado registro por meio da seleção na listagem de itens. Esta atividade resultou na abertura do registro. Após a verificação dos valores em tela ao qual se constatou não ser o registro esperado, utilizou-se do elemento de interface *cancelarBtn* para ter acesso novamente a listagem de itens.

Uma vez nesta listagem, utilizou-se do filtro de busca por meio da entrada de dados no elemento de interface *searchEditTxt*. O dado informado como critério do filtro foi *Unesc*. No retorno dos resultados obtidos verificou-se que apenas um registro possuía o nome desejado. Por último, encerrou-se a execução das atividades e gerou-se o caso de teste automatizado.

```
@Test
public void teste() throws Exception {

    searchEditTxt.clear();
    getElementUsingParentIdAndTextAndScrollTo("list", "Unesc").click();
    Assert.assertEquals(false, elementHasFocus(nomeMotorista));
    Assert.assertEquals("Unesc", nomeMotorista.getText());
    getElementByIdAndScrollTo("nomeMotorista");
    getElementByIdAndScrollTo("cancelarBtn");
    cancelarBtn.click();
    Assert.assertEquals(false, isElementDisplayed(empty_text));
    if(!elementHasFocus(searchEditTxt)){
        searchEditTxt.click();
    }
    Assert.assertEquals(true, elementHasFocus(searchEditTxt));
    searchEditTxt.setValue("Unesc");
    Assert.assertEquals("Unesc", searchEditTxt.getText());
    Assert.assertEquals(false, isElementDisplayed(empty_text));
}
```

Figura 1 – Caso de teste automatizado gerado partir do registro das atividades

Na figura 1 pode-se observar o caso de teste automatizado gerado a partir do registro das atividades do usuário durante a etapa de execução do cenário estudado.

A análise do caso de teste gerado indica uma aplicação bem-sucedida da pesquisa mediante ao cenário exposto, uma vez que instruções do algoritmo de teste automatizado estão em conformidade com as ações executadas pelo usuário. Ou seja, a pesquisa foi capaz de assemelhar um caso de teste automatizado ao contexto e forma de utilização da aplicação móvel feita por um usuário.

Contudo, verificou-se que a asserção de quantidade de registros encontrados pelo filtro de busca não está presente no caso de teste automatizado. Isto se dá ao fato da instrução de registro desta atividade não ter sido inserida no código da aplicação de testes.

Entende-se por meio deste resultado que a geração do caso de teste automatizado depende diretamente do uso correto dos métodos de registro de atividade. A omissão de um comando, ou sua utilização de forma incorreta pode refletir na geração de casos de teste que não representam as atividades executadas pelos usuários.

Em paralelo a criação deste caso de teste obteve-se a documentação do teste gerado. Esta documentação especifica os passos executados durante a interação do usuário com a aplicação móvel.

```
Documentação do caso de teste
----------------------------
Passo 1: Limpar o conteúdo do elemento 'searchEditTxt'
Passo 2: Selecionar o item 'Unesc' no elemento 'list'
Passo 3: Verificar se o elemento 'nomeMotorista' está sem o foco
Passo 4: Verificar se o elemento 'nomeMotorista' possui o valor 'Unesc'
Passo 5: Acessar a tela 'Cadastro de Motoristas'
Passo 6: Clicar no elemento 'cancelarBtn'
Passo 7: Acessar a tela 'Listagem de Motoristas'
Passo 8: Verificar se o elemento 'empty_text' está oculto
Passo 9: Focar no elemento 'searchEditTxt'
Passo 10: Verificar se o elemento 'searchEditTxt' está com foco
Passo 11: Digitar o texto 'Unesc' no elemento 'searchEditTxt'
Passo 12: Verificar se o elemento 'searchEditTxt' possui o texto 'Unesc'
Passo 13: Verificar se o elemento 'empty_text' está oculto
```

Figura 2 – Documentação do caso de teste do primeiro cenário

Pode-se considerar válida a documentação ilustrada na figura 2 dado ao fato de que as atividades executadas estão documentadas de forma que possibilite uma possível consulta e reprodução das atividades.

3.1. Discussão dos resultados

Afirma-se por meio da execução deste estudo que se obteve uma melhoria na criação dos casos de teste automatizado, alcançando os resultados esperados.

Determinou-se a execução desta pesquisa como bem-sucedida por meio do cumprimento de requisitos qualitativos referentes aos casos de teste automatizado gerados. A partir da análise realizada constatou-se as características que indicam a melhoria na criação de casos de teste automatizado (GARRETT, 2012, tradução nossa):

a) capacidade de reproduzir as atividades dos usuários por meio de teste automatizado;

b) fidelidade entre a especificação dos casos de teste automatizado com o contexto de utilização dos usuários;

c) aumento da cobertura dos casos de teste automatizado e aumento da capacidade de revelar defeitos;

d) aprimoramento da característica exploratória dos casos de teste automatizado indicando diversificação dos passos contidos nos casos de teste automatizado.

Observa-se por meio da análise deste caso especifico, que não se descarta a possibilidade de aplicação deste estudo em outras aplicações móveis. Acredita-se que a execução bem-sucedida desta pesquisa em outras aplicações móveis poderá proporcionar os mesmos benefícios resultantes neste estudo.

Constatou-se também, por meio dos resultados obtidos, que a eficiência desta pesquisa em gerar casos de teste automatizado e ampliar a cobertura de testes exploratórios tende a crescer a medida que o número de usuários da aplicação móvel aumenta. Isto ocorre dado ao fato de que as atividades executadas por cada usuário diferem entre si. Consequentemente há um aumento na diversidade dos cenários e das atividades utilizadas como base para geração de novos casos de teste.

Observou-se durante os estudos iniciais sobre algoritmos de teste automatizado que a especificação manual de casos de teste automatizado exige um nível de esforço elevado. Isto foi possível por meio da comparação do tempo investido para criação dos de teste automatizado e utilização da aplicação deste estudo. Constatou-se uma redução considerável no tempo necessário para especificação de casos de teste automatizado. Esta diminuição no tempo investido para especificação dos casos de teste automatizado possibilita a equipe de testes realizar aprimoramentos no processo de garantia de qualidade de software.

4. Conclusão

O desenvolvimento e a aplicação dos algoritmos de registro de atividades e geração de caso de teste automatizado proporcionaram a experimentação de recursos e estratégias que auxiliam o processo de especificação do teste automatizado.

Conclui-se por meio dos da análise dos resultados, que houve contribuição na diminuição do tempo necessário para especificação de casos de teste automatizado.

Pode-se também considerar como uma vantagem desta pesquisa a possiblidade de participação de equipes externas na criação de casos de teste automatizado, incluindo usuários participantes de testes de aceitação, usuários finais em ambiente de produção e equipes de testes de software. Isto se dá ao fato da geração dos casos de teste ocorrer de forma transparente para os usuários e pode ser realizada por meio da própria aplicação móvel.

Conclui-se por meio da aplicação desta pesquisa, que a cobertura e a qualidade dos casos de teste exploratórios depende de diversos fatores, que variam desde a experiência do testador em especificação de casos de teste até o conhecimento sobre as regras de negócio da aplicação. Estes fatores possuem influência direta na quantidade e na qualidade dos testes exploratórios.

5. Referências

BARTIÉ, Alexandre. **Garantia da Qualidade de Software.** Rio de Janeiro: Elsevier Editora Ltda., 2002. Disponível em: <https://books.google.com.br/books?id=O57Us2kUh4oC&printsec=frontcover&hl=pt-BR#v=onepage&q&f=false>. Acesso em: 25 set. 2017.

BASTOS, Anderson et al. **Base de conhecimento em teste de Software.** São Paulo: Martins, 2007. p.35-70

BERNARDO, Paulo Cheque; KON, Fabio. **A Importância dos Testes Automatizado**s. Artigo publicado na Engenharia de Software Magazine, 2008. Disponível em: <https://www.ime.usp.br/~kon/papers/EngSoftMagazine-IntroducaoTestes.pdf >. Acesso em: 2 nov. 2017.

GARRETT, Thom. **Useful Automated Software Testing Metrics.** 2012. Disponível em: <http://idtus.com/wp-content/uploads/2012/09/UsefulAutomatedTestingMetrics.pdf>. Acesso em: 03 maio 2018.

MORITZ, Evelyn. **Case study: How analysis of customer found defects can be used by system test to improve quality**. In: INTERNATIONAL CONFERENCE ON SOFTWARE ENGINEERING - COMPANION VOLUME, 31**.** Vancouver: Ieee, 2009. p. 123 - 129. Disponível em: <https://ieeexplore.ieee.org/document/5070970/>. Acesso em: 04 jun. 2017.

PFALLER, Christian et al. Multi-Dimensional Measures for Test Case Quality. **2008 Ieee International Conference On Software Testing Verification And Validation Workshop**, [s.l.], p.1-4, 2008. IEEE. http://dx.doi.org/10.1109/icstw.2008.28. Disponível em: <https://ieeexplore.ieee.org/document/4567035/>. Acesso em: 07 jun. 2018.

PRESSMAN, Roger S.; MAXIM, Bruce R. **Engenharia de Software:** Uma Abordagem Profissional. 8. ed. Nova Iorque: Mariana Belloli, 2016. 968 p.

SOMMERVILLE, Ian. **Engenharia de software**. PEARSON BRASIL, 2011. ISBN 9788579361081. Disponível em: <https://books.google.com.br/books?id=H4u5ygAACAAJ>. Acesso em: 07 maio 2017.